TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG

The University of Resources. Since 1765.

Fakultät für Mathematik und Informatik
Institut für Informatik
Lehrstuhl für Virtuelle Realität und Robotik

**Bachelor Thesis**

# aMoSeRo - a Mobile Sensor Robot

**Paul Petring**

Applied Informatics
Specialization: Economy

Matriculation register: 48406

October 13, 2014

Tutor/First Proofreader:
M.Sc. Erik Berger

Second Proofreader:
Prof.Dr. Jung

# Contents

**1 Introduction** . . . **4**
1.1 Mobile robots . . . 4
1.1.1 iRobot . . . 4
1.1.2 Boston Dynamics . . . 7
1.1.3 Willow Garage . . . 10
1.1.4 University projects . . . 12
1.2 Problem description . . . 14
1.3 Structure . . . 14

**2 Software** . . . **15**
2.1 Requirements . . . 15
2.1.1 Software Reuse . . . 15
2.1.2 Distributed computation . . . 16
2.1.3 Robotic Software Environments . . . 16
2.1.4 Licenses . . . 17
2.2 Robot Application Frameworks . . . 18
2.2.1 Microsoft Robotics Developer Studio *(MRDS)* . . . 18
2.2.2 Cyberbotics - Webbots . . . 18
2.2.3 Player Project . . . 19
2.2.4 Robot Operating System *(ROS)* . . . 19
2.2.5 Conclusion and consequences . . . 22

**3 Hardware** . . . **23**
3.1 Requirements . . . 23
3.1.1 Low cost . . . 23
3.1.2 Software conditions . . . 24
3.1.3 Physical properties . . . 24
3.2 Modular design . . . 25
3.2.1 Sensors . . . 25
3.2.2 Accumulators . . . 31
3.2.3 Processors . . . 32
3.2.4 Actuators . . . 36
3.3 Further extensibility . . . 40

**4 Implementation** . . . **41**
4.1 aMoSeRo Hardware . . . 41
4.1.1 List of parts . . . 41
4.1.2 Case . . . 42
4.1.3 Wiring schematic . . . 43
4.1.4 Combination . . . 44
4.2 aMoSeRo ROS package . . . 44
4.2.1 amosero_description . . . 44
4.2.2 amosero_bringup . . . 45
4.2.3 amosero_teleop . . . 45

4.2.4 amosero_navigation . . . 46
4.2.5 amosero_viz . . . 46
4.3 aMoSeRo Microcontroller . . . 46

**5 Evaluation 47**
5.1 aMoSeRo 2D Laser scan frame rates . . . 47
5.2 aMoSeRo 2D SLAM . . . 48
5.3 aMoSeRo 3D PointCloud Maps . . . 49

**6 Conclusion 50**

# 1 Introduction

In our society the abilities of robots range from fulfilling repetitive tasks in industry over vacuum cleaning service robots through autonomously operating in extreme environments. The complex systems originating from electronics, informatics and mathematics therefore are found from the bottom of the sea to space and are continuously improved by new technologies and further research. Nevertheless, they come with one big disadvantage; when their capabilities go beyond basic tasks and reach localization, mapping or planning, they tend to be really expensive. This fact leads to a dramatic economic damage in cases of losing an advanced robot, which is why their real usage fields often get limited.
This essay discusses a new approach for creating low cost mobile robotic platforms. Therefore, we will now introduce the corresponding terminologies.

## 1.1 Mobile robots

The word *robot* was first introduced by Karel Čapek in 1921 within his book Rossum's Universal Robots in which he describes a factory creating artificial people for working purposes called the *roboti* coming from the Czech word *robota* meaning *forced labour* [Dic].
Nevertheless, a *robot* in this thesis is defined following the *Oxford Dictionary* as a mechanical, virtual or artificial machine that is capable of carrying out a complex series of actions automatically, especially one programmable by a computer [Pre10].
The next term that must be examined is *mobile*. Following the common definition, a device with the capability to perform motion or to operate during motion can be considered to be mobile. Furthermore, *motion* itself can be described as the changing of position according to a reference point in function of time, consisting of the physical units displacement, direction, velocity, acceleration and time [Nav14]. Consequently, *mobile* in a robotic context are devices that are able to move by themselves and not fixed to one physical position or location. Under this definition, also robotic arms like they are used in industrial factories are considered mobile robots.
The wide range of mobile robots makes it necessary to specify the topic of the thesis further so that we have a closer look at *Unmanned Ground Vehicle*s (UGVs) and *Automated Guided Vehicle*s (AGVs), which are able to move on most environments mostly using propulsion systems like wheels or tracks. By that other robots like flying devices or autonomous underwater vehicles can not be discussed in detail as they have other specifications and therefore properties which would unfortunately go beyond the scope of this bachelor thesis.
In order to find the main characteristics of UGVs and AGVs we now examine some of the most important existing robotic ground vehicles and their manufacturers more closely. Starting by one of the most successful civil and military industrial manufacturers, over a non-profit open source approach think tank and finally coming to an excerpt of the university context.

### 1.1.1 iRobot

Still one of the most successful companies by the metric of sold service robots in the domestic sector and therefore the most popular robot building one today is called *iRobot*.
Its original name has been *Artificial Creatures Inc.* and soon changed with one of the first robots called the *iRobot LE* [iRo90], where the *i* was an acronym to *internet* and not like

today's interpretations are referring to Asimov's *I, Robot* [Lin14]. During that period *iRobot* followed behavior control and swarm implementations over the new medium.
Initially the pioneer in the robot industry, *iRobot*, got co-founded in 1990 by the entrepreneurs Rodney Allen Brooks, Colin Angle and Helen Greiner which all have met as members of the *Massachusetts Institute of Technology* (MIT) *Computer Science and Artificial Intelligence Laboratory* (CSAIL).
The former Panasonic professor at MIT Brooks has been popularizing the actionist approach of robotics or in other words has been one of the entrepreneurs of behavior based robotics.
With a master's degree in computer science and a bachelor's degree in electrical engineering, Colin Angle realized one of the first projects of the company called *Genghis*, a six legged robot as a part of his thesis. Later he led the development of behavior-controlled rovers for NASA which lead to the mars rover Sojourner in 1997.
Whereas Hellen Grainer, also master of computer science and owning an additional bachelor's degree in mechanical engineering soon took over the business sector of the corporation. She also led *iRobot* into the military market place [CyP14]. Grainer is a laureate with numerous awards like Pioneer Award from the *Association for Unmanned Vehicle Systems International* (AUVSI) in 2006 and is a member of the *Open Source Robotics Foundation* (OSRF) [OSR14].
To solve the demand of new robot engineers, in 2009 *iRobot* initiated the *Starter Programs for the Advancement of Robotics Knowledge* (SPARK) as a part of the *Science, Technology, Engineering and Math* (STEM) education initiative, which supports schools to teach robotics.

## Service robots

'A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations' in ISO 8373:2012 [IFR12]. By means of this definition by the *International Federation of Robotics* (IFR) *iRobot* created multiple robots, from which we examine the most popular ones in more detail.

**Fig. 1:** the iRobot Roomba 880 [Rob14f]

**Fig. 2:** the iRobot Scooba 450 [Rob14g]

**Fig. 3:** the iRobot Create [Ima14]

**Roomba** is a family of robot vacuum cleaners. After each cleaning cycle a Roomba returns to its docking station to recharge. Consequently it reached a high level of customer value, which is one of the main reasons that different versions of the autonomous service cleaners have already sold about 10 million units since 2002 [iCi14b]. Therefore the Roomba is still one of the most successful robot series.
We introduce *Depth Sensors* in section 3.2.1 in detail, but it is important to mention that current versions of the Roomba use cheap cameras instead of expensive technologies like *light amplification by stimulated emission of radiation* [Dic14]s (Lasers) to perform *visual*

*Simultaneous Localization and Mapping* (vSLAM). However, a current Roomba still costs about 600 to 700 USD at local distributors.

**Scooba** is a family of floor mopping domestic robots and the follower of the iRobot Braava [iCi14c]. The Scooba series was released with limited quantities in late 2005 before a full product release in 2006. It cleans floors with up to 300 square feet in three basic steps: first it sweeps loose dirt and pre-soaks the area by leaving a thin sheen of soaped water. Next time the Scooba crosses the same spot he scrubs with 600 *rounds per minute* (rpm) and vacuums dirty water off the floor. Finally it squeegee-finishs the spot from remaining dirt. Path finding algorithms require all three stages to occur at the same time, which among other things like refilling the robot after a finished cleaning cycle reduces its customer value.
Following the *Roomba* price philosophy, a current Scooba also costs about 600 USD plus the required drying dock with additional 100 USD.

**Create** After the first units of the Roomba were released technique enthusiasts soon tried to control it by custom software which led *iRobot* to release an *application programming interface* (API) for the serial communication [Eng14]. For that reason the serial port was made easily accessible to make modifications easy to perform by the *iRobot Roomba Open Interface* (ROI) [Wir10b]. In 2007 the company went one step further and created a new robot based on the *Roomba* 400 series. They explicitly designed the so called Create for robotics development by removing the cleaning abilities and replacing it by a cargo bay that can hold a usual *Single Board Computer* (SBC). It includes an embedded computer ready for serial communication. As a consequence, all computers including laptops with a serial communication device are able to run the robot. Soon many robotic frameworks implemented the API and subsequently are able to control the real world robot.
In 2010 a student at MIT's Personal Robotics Group called Philipp Robbel used the Create to realise a low cost mobile robot with a *Microsoft Kinect* [Wir10a]. This project has been part of the *Create Challenge* where *iRobot* was offering 5000 USD with the goal of creating an 'innovative robot that's functional, helpful, entertaining, whimsical or simply amazing' [iSu07].
Though the poor availability of the *Create* the price varies heavily between 130 EUR [Zag14] and 300 EUR online.

### Military robots

Broadly defined, military robots can be found since the soviet teletanks and the Goliath teleoperated mobile mine, which have been created since World War II. Since then, numerous unmanned vehicles and airplanes have been created for military purposes and therefore are considered as military robots in this thesis.
Already in its early stages *iRobot* was researching in the military sector. Ariel and Fetch were the first two UGVs to detect and dispose land mines both on ground and sea, but the most successful so far with about 4500 sold devices is the multipurpose mobile robot *PackBot.*

**PackBot** is a series of military AGVs that e.g. have been used in Fukushima nuclear plant after the 2011 earthquake following tsunami and the World Trade Center on 9/11.
The *PackBot* as shown in **Fig.4** was developed after winning a *Defense Advanced Research Projects Agency* (DARPA) contract in 1998 [iRo14a] and has led to the current base model the PackBot 510.

This mobile robot can be used in many different scenarios because of its ability to climb stairs, role over rubble, traverses rocks, mud and snow at speeds up to 5.8$mph$. Furthermore it has a modular design by providing several cargo bays. Those can be equipped with a variety of sensor boxes, which enables the PackBot to be used for a lot of tasks like searching buildings, disposing bombs or surveillance purposes. In case that a PackBot flips during an operation, it is capable of self-righting.

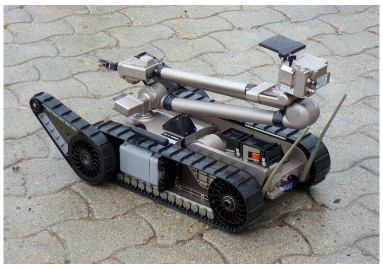


**Fig. 4**: the iRobot PackBot 510 E.T [WiM14]

A prerequisite for running the PackBot is a so called *Operator Control Unit* (OCU), which is a special 15 inch outdoor laptop. The data sheet of the PackBot [iCi14a] also reveals that it can be run up to four hours or 10 miles of travel with two batteries. The base system weights 10.89$kg$ without extensions like arms, cameras or accumulators.
Another interesting fact to mention is the PackBots ability to communicate by 4.9$ghz$ mesh networking - in a simple setup between robot and OCU.

**Negotiator** is an AGV that was developed based on the more powerful *PackBot* in 2008.
As a lower cost version of the *PackBot* it was designed to be more affordable by law enforcement, firemen and other public safety officials to replace humans in dangerous situations. Examples are burning houses, contaminated areas or negotiations where operators are required to evaluate dangerous situations from a safe distance [tJ08].

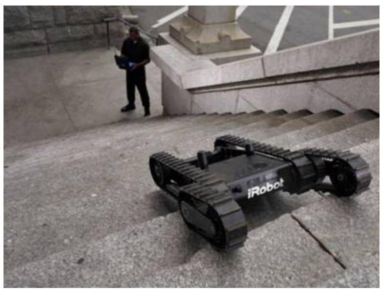


**Fig. 5**: the iRobot Negotiator [iRo14b]

The slimmed down robot, as illustrated in **Fig.5**, is equipped with a color video and low light infrared camera which gets supported by an additional infrared LED array for illuminating dark surroundings. It allows several additional modules like a day/night pan and tilt camera systems, flood lights or a *MultiRAE Plus* gas monitor detection system.
The two nickel metal hydride (NiMH) batteries are capable of providing between three to six hours of runtime depending on usage and allow a rapid charging system which charges faster than comparable lithium-ion batteries. This enables the robot to move at 3.1$mph$.
The entry price point can be estimated around the 20 000 USD mark. [Giz08]

### 1.1.2 Boston Dynamics

*Boston Dynamics* is a high-tech research company that was created as a spin-off from MIT by the National Academy of Engineering member Marcus Raibert in 1992. As a pioneer of legged dynamics and locomotion in robotics, he and his colleagues showed that it is possible to balance one, two or four legged machines to be stable and furthermore to be able to adapt to special environments very well.
In 1980 Dr. Raibert originally created the Leg Lab, a research laboratory to explore walking machines at Carnegie Mellon University. After that, he moved to the laboratory to MIT before starting his own company with simulation software for the US military.
By his definition only half of the earth's landmass is accessible to existing wheeled and tracked

vehicles, which is why *Boston Dynamics* mostly develops multiple legged robots including humanoids like PETMAN or animal based ones like *BigDog*, LS3 or Rise. The company also did some important progress in the field of AGVs with the *RHex* or the *SandFlea* by adding animal like behaviour and features.
Most of their projects are funded by DARPA or other military facilities. Since 13 December 2013, when the company was acquired by Google, new fields of usage like a combination of the Google Car Project and PETMAN for fully automated delivery of packages are targeted. Consequently, today, the company is managed among other robotic companies acquired by Google by Andy Rubin, who is so far known as the inventor of the smartphone operating system Android [NYT13].

## BigDog

In modern warfare, soldiers need to carry more and more equipment during operations in rough terrain. Currently it is unavoidable to use mules or donkeys which are able to carry one third of their body weight, but they tend to be difficult to handle. Furthermore the animals are lacking in features like being able to be sent ahead or jump off from an aircraft. Additionally they mostly require attention even if they are not used, which among other things means that it is not possible to store them in warehouses. [Rai10]
In 2005 *Boston Dynamics* created a four legged mobile robot called *BigDog* which is able to climb slopes up to 35 degrees. *BigDogs* development already started in 2003 in partnership with the British robot maker Foster-Miller, NASA's Jet Propulsion Laboratory and Harvard.
Powered by a go cart engine and moved by a hydraulic pump like they are used in aircrafts for the last 30 years, the *BigDog* in **Fig.6** is a twenty joints mobile system. Each leg consists of four back drivable hydraulic actuators, a powerful servo motor and a spring to passively adjust balance. The usage of hydromechanical principles allow a high power to weight ratio of about 1200 Watts to two pounds. That makes the usage of gears obsolete and directly pushes the power to the floor.

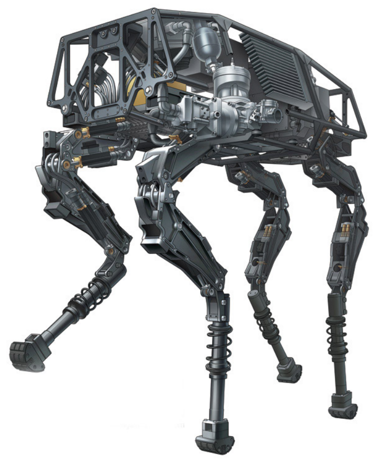


**Fig. 6**: the Boston Dynamics *BigDog* [Qwr14]

The main issue of a legged robot is its balance. For solving that task, Dr. Raibert and his team divided the problem into three parts. First, providing the counter reaction to gravity by the legs that are currently in contact to the ground called *Support*. Next, the stopping to tip over during the locomotion process, while center of mass is not in center of the robot's body using inverted pendulums as internal sensors called *Balance*. Finally, the task of keeping the body of the robot at a desired place or in a special position during external influences like foot slipping or lateral velocity - called *Posture*.
Because all three problems are solved internally with a high level of autonomy, it is possible to easily control the robot with a single remote controller. This also makes it possible to use higher level algorithms for navigation like following squad leaders can be successfully implemented. An important prerequisite for the functional balance system is good environment awareness achieved by a sensor system. Therefore the *BigDog* consists of external sensors like GPS, a ring laser, a gyroscope, a *Light radar* (Lidar) and sensors for stereo vision. Further-

more, internal sensors for values like hydraulic pressure flows, engine temperature, forces and angles around the joints are required. In total the *BigDog* has about 50 sensors depending on the final setup. Additional controls are needed for problems like estimation of odometry by the legs in support or traction control to avoid and recover from foot slips. Additionally, reacting to terrain disturbances in the stepping cycle or swinging legs to avoid collisions are essential. When responsibly adjusting the engine power the *BigDog* needs about two gallons of gasoline per mile. By additional tanks with up to 450 pounds of weight this allows a maximum range of 12.8 miles. The total payload the *BigDog* can carry heavily depends on the terrain and lies between 100 pounds on trial terrain and 240 pounds on flat. At the moment the system has three movement modes: walking at one miles per hour (mph), trotting at three mph, and jogging at up to six mph.
There is a current project for a successor called *Legged Squad Support System* (LS3) funded by DARPA and the United States Marine Corps. It shall fulfill higher demands in hardware like greater ranges and run time or avoiding the loud motor noise that is emitted. *Boston Dynamics* experimented with electronic motors inside the *BigDog* and was faced with a run time of only ten minutes and a battery consuming the total carry capacity [Rai10].
One problem of the *BigDog* is the lack of self righting - in case the robot gets on the ground like lying on its side, it is presently not able to get up again without the help of an operator. This should be fixed by the LS3 with some adjustments in hardware design like the usage of shoulders. There are already successful simulations of LS3 getting up from the back by swinging the legs and building momentum. Furthermore, the LS3 will improve the *BigDog* at redundancy by additional backup systems. [Rai10].

### SandFlea

The *SandFlea* was created by a cooperation between *Boston Dynamics* and *Sandia National Labs* in 2009 [IEE12]. The UGV was originally called the *Precision Urban Hopper*, but was renamed by the DARPA and the *Rapid Equipment Force* (REF).
The *SandFlea* originates from a class of *Insecta*, caused by its *CO2* powered piston in the back [IEE09] which enables the eleven pounds robot to jump adjustable heights between one and nine meters. Therefore, it is more useful than previously used small airborne systems which are more expensive and less reliable. The four unusual shaped wheels act as shock absorbers for hard landings. It is furthermore used on squad level as an optical system that finds its ways behind enemy lines during operation and has been already in combats in Afghanistan.

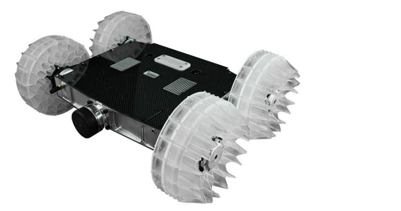


**Fig. 7**: the Boston Dynamic SandFlea [Qwr14]

The *SandFlea*, like shown in **Fig.7**, is 33*cm* long, about 46*cm* wide and has a height of 15*cm*. The endurance of its batteries enables the robot to run up to two hours and do 25 jumps per charge. It is furthermore built to tolerate humidity, salt, oil and extreme sand environments.
The resolution of the video camera is 320x240 pixels and has a maximal still image resolution of 1280x960 pixels.

### RHex

The *Boston Dynamics RHex* [Dyn12] is a realized hexapod design following the innovations done by multiple universities funded by the DARPA.
Numerous papers came to the conclusion that the hexapedal approach has a better locomotion than wheel or track-based systems [D.E01] but leaving the potential of legged robots behind. The difficult terrain robot consists of a rigid body with six compliant legs with one independently actuated *Degrees of Freedom* (DOF) per leg.

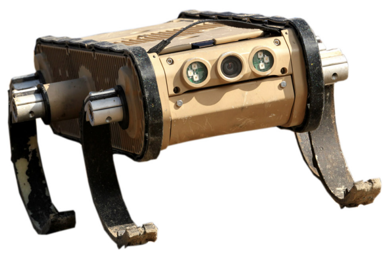


**Fig. 8**: the Boston Dynamic RHex [Qwr14]

The data sheet of the *Boston Dynamics RHex* [Dyn13] discloses some additional details of the 12.5$kg$ robot like its special LiIon batteries mark *BB2590* which allows run times of up to six hours at two mph on natural terrain while being able to climb up slopes up to 60 percent in slope walk or even higher in up to 84 percent climb mode. Another point to mention is the IP radio which has the very high range between 400 and 700$m$. Moreover the *RHex*, as illustrated in **Fig.8**, is IP67 sealed and consequently water submersible.

## 1.1.3 Willow Garage

Scott Hassan, the founder of *Willow Garage*, was a doctoral computer science student at the *Stanford University* in 1996, where he has met both Larry Page and Sergey Brin, the later founders of *Google*. Hassan helped them developing the google source code what Larry Page thanked him later by giving him a big amount of *Google* shares. After additionally selling his own company *eGroups* - an email group managing software - to *Yahoo* for 412 million USD Hassan's wealth today can be estimated around one billion US dollar [Ols08].
In late 2006 the open source and free software enthusiast Hassan created the robotics think tank called *Willow Garage* in *Menlo Park*, *California* - outside of *Silicon Valley*, as the newly founded company had no ambition to make money.
The company probably got most famous for the creation of the *Robot Operating System* (ROS), an open source solution for a lot of robotic problems, which we explain more detailed in section 2.2.4 *Robot Operating System* (ROS).
*Willow Garage* supported multiple spin-off-companies. Most important the autonomous robot producer *Robotnik* [Rob14e] and *Industrial Perception Inc.* (IPI), which builds guiding robots for automated material handling at commercial distribution centers [IPI14].
Beginning with February 11th 2013 *Willow Garage* changed to a self-sustaining company [WG14g] which led to the majority of *Willow Garage* Inc. employees moving into Hassan's newly founded company called *Suitable Technologies* [Tec14] that today sells the telepresence robot *Beam*.
Since January 17th 2014 the longstanding Canadian partner company *Clearpath Robotics* [Cle] took over the support and service responsibilities for the leading mobile manipulation platform *Willow Garage* PR2 [WG14h].
In the following, we introduce some of their mobile robots.

### TurtleBot

The *TurtleBot* is a series of robots created by Tully Foote and Melonee Wise for *Willow Garage* [Ack13] around 2010. Shortly after *iRobot* released their first domestic robots they moved away from their first attempts of creating a low-cost ROS robot with the *Lego NXT* [Min14] (which was too weak on computational power and expandability) over *iRobot Roomba* to finally the *iRobot Create*. The main advantages of these are that they make use of the recently released *Microsoft Kinect* and generate a better odometry by adding gyroscope. Furthermore, they created an open hardware case that is suitable for most indoor applications. After leaving the development stage the *TurtleBot*, which has been named after the *Xerox Parc Turtle Graphics* by Seymour Papert [SP76], has become a very valuable platform for education and research. Tully Foote now works for the OSRF [OSR14], the current holder of the trademark *TurtleBot*. Melonee Wise left *Willow Garage* and founded her own company called *Unbounded Robotics* [Rob14b] which focuses on creating low cost robots for research and education and up to now created the robot *UBR-1* [Rob14c].

**TurtleBot 1** Mobile computer systems beside laptops have not been very popular yet and therefore are not as cheap as average laptops, so the inventors decided against a screenless robot and used an *ASUS 1215N* (Intel Atom D525 Dual Core Processor with 2GB RAM [WG14c]) laptop as main processing unit.
Its mobile base, an *iRobot Create*, allows an additional load capacity of $5kg$ to the $5kg$ of the robot itself with a speed of about $0.65m/s$ ($2.34km/h$). The 3000 mAh NiMH battery pack which is located inside the mobile base lasts around 1.5 hours. Whereas the *Microsoft Kinect* requires an additional $12V$ $1.5A$ power supply which can be activated by software.
It is possible to build your own *TurtleBot 1* like it can be seen in **Fig.9** by instructions given on *Willow Garage*'s website, including all required parts of the case as printable three dimensional (3D) models [WG14a]. Currently, the *TurtleBot 1* costs between 1000 and 1200 EUR at local distributors.

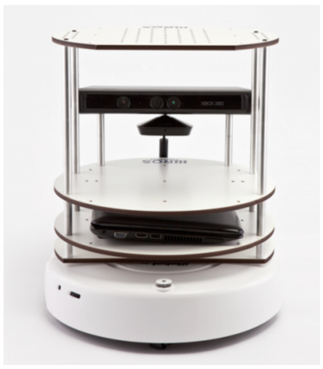


**Fig. 9**: the Willow Garage TurtleBot 1 [WG14f]

**TurtleBot 2** After the release of *TurtleBot 1* the feedback of the robotic community was enormous. As Tully Foote described, Daniel Stonier from *Yujin Robotics* simply wrote an email suggesting a cooperation between *Yujin* and *Willow Garage*. They exchanged a wish list of what an ideal mobile robot base should be capable of and this led to the development of the *iClebo Kobuki* base. It has many advantages over its predecessor: improved odometry measurement precision, the usage of an open protocol and greater autonomy by better charging stage handling. Furthermore it provides a greater general load and higher speed and brings better mobility by a larger diameter of its wheels. This enhanced the capacity to overcome obstacles up to $12mm$ and is the reason why it is able to turn slightly faster with $180deg/s$.
Therefore, the *TurtleBot 2*, like he is illustrated in **Fig.10**, gained $10cm$ of height and $1.3kg$ of weight while keeping the maximum additional load at $5kg$.

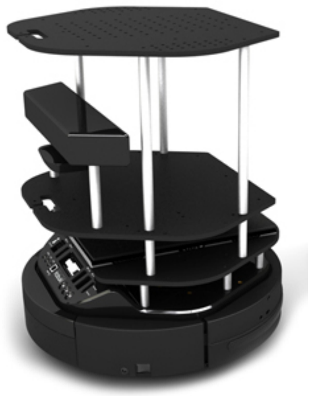


**Fig. 10**: the Willow Garage TurtleBot 2 [WG14d]

The battery and power system varies between a standard $14.8V$ $2200mAh$ Li-Ion ($3h$) and an extended $4400mAh$ Li-Ion ($7h$) battery.

**TurtleBot 3** The *TurtleBot 3* 3 has not yet been created. As both inventors do not work for *Willow Garage* anymore, the concept might change significantly. But when they were asked for speculations both agreed on the implementation of a 360-degree Laser, more powerful batteries and even better obstacle traversal. This might influence later decisions according to our own robot.

### Husky

With 1000 sold units until 2014, the *Husky* is the flagship UGV of *ClearPath Robotics*. In 2009, the company was founded by four graduated students from the *University of Waterloo's* mechatronics engineering program. The company is very closely connected to *Willow Garage* and therefore the *Husky* is a very good example for UGVs running their ROS. Since then the company grew to more than 50 employees selling nine different robot models.
The $99cm$ long, $67cm$ wide and $39cm$ high comparatively big mobile base has a weight of $50kg$ and can carry additional payload of $75kg$ at a maximum speed of $3.6km$ per hour. It carries a battery with a $20Ah$ capacity that allows a total run time of up to three hours. Furthermore *ClearPath Robotics* later released a bigger version called the *Grizzly*.
Its properties make the *Husky* an ideal platform for a wide variety of tasks which reach from mining applications up to the Canadian Space Agency or NASA's *HI-SEAS* - a long term program for Mars missions. Its current price is about 70 000 USD in an equipped version.

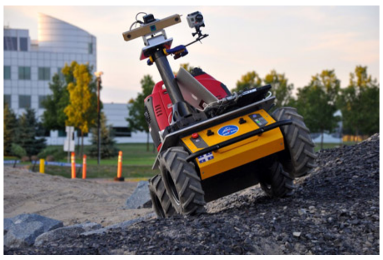


**Fig. 11:** the Clearpath Robotics Husky [Ack12]

### PR2

The Robotic Research platform *PR2* [WG14e] is the state of the art product of *Willow Garage*. The 50 world wide existing *PR2*s, as shown in **Fig.12**, are created to break the wheel reinvention in robotics and try to enable researchers to easily reproduce code in papers.
Therefore it combines an omnidirectional base with two five DOF arms and multiple sensors, including laser scanners, cameras, inertial measurement units and many more [WG14b]. On the software side it is run by *Robot Operating System* (ROS).
Different versions of the very modular *PR2* are currently used by 34 institutions in 12 countries that form a very active community [PR214a]. The prices of a base *PR2* for educational institutions start at 280,000.00 USD excluding taxes and shipping [PR214b].

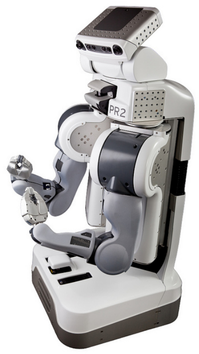


**Fig. 12:** the Willow Garage PR2 [Raz11]

## 1.1.4 University projects

Another essential branch of robotic development are university projects. They provide fundamental research and are the root of every commercial robot company.
Unfortunately there are more projects existing than a single thesis can cover. In the following, we therefore introduce two projects that are somehow connected to this work. We are starting with the *Carnegie Mellon University*, the place where Dr. Raibert started with the *Leg Lab*

and one of the rare places of robots that are explicitly developed in relation to mining. The second one is the *Technical University Darmstadt* which has been the first German winner of the Rescue Robot League in 2014 [Rob14a].

### Carnegie Mellon University - Groundhog

The *Carnegie Mellon University - Groundhog* is a four-wheeled UGV built in 2004. First experiments already took place in 2002 [CMU02] and relied on cable communication.
It has been developed to explore and map subterranean spaces like found in abandoned coal mines. It performed nine autonomous missions documented by students of the Robotics Institute's Mobile Robot Development class of Carnegie Mellon University. Under the lead of Scott Thayer and the *Fredkin University* professor William L. "Red" Whittaker the robot has been developed as a response to an incident at the *Quecreek Mine*, where nine miners nearly died after an accidentally breached wall, which caused the flooding of the mine. One of the main issues back then was missing map data.

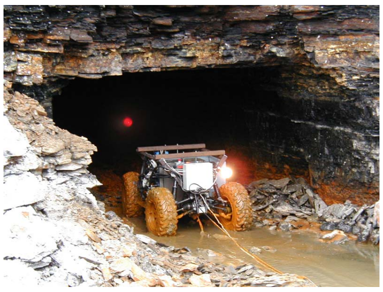


**Fig. 13:** the Carnegie Mellon Groundhog [CMU05]

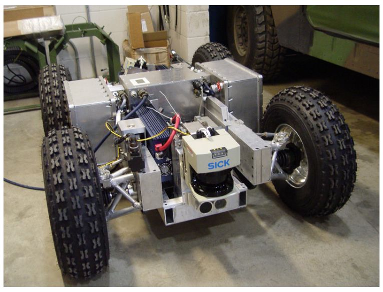


**Fig. 14:** Carnegie Mellon CaveCrawler
**Source:** Carnegie Mellon University [CMU14]

The *Carnegie Mellon University - Groundhog*, like illustrated while one of its first underground experiments in **Fig.13**, is capable of traveling at 1.6$km$ per hour and is equipped with an array of cameras as well as gas, tilt and sinkage sensors. Furthermore, a laser scanner and a gyroscope enables this early UGV to use *Simultaneous Localization and Mapping* (SLAM) algorithms.
"The Groundhog is only the beginning. We see future generations of machines that will swim, crawl and climb through mines to enhance safety, support rescue and ultimately enable robotic operations beyond mining in caves, bunkers, aqueducts and sewers." Whittaker [fCMUC14]

### TU Darmstadt - Team Hector

The *Heterogeneous Cooperating Team of Robots* (Hector) results from a PhD program of *Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments* of the Technical University Darmstadt [Dar14]. The main task of Hector is the development of navigation and coordination algorithms for multiple autonomous vehicles like shown in **Fig.15**. They share the created software under open source licenses [GRK14]. Hector has been very successful at Robotic Conventions and Tournaments like the *RoboCup Rescue* [Res14] by using funds of the *Deutsche Forschungsgemeinschaft* [DFG14] (DFG) to use expensive technologies like Laser scanners or Lidars.

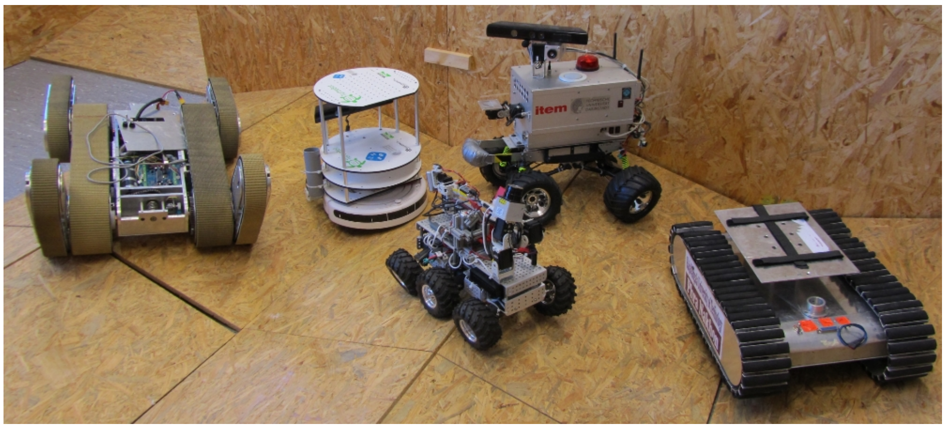


**Fig. 15**: the TU Darmstadt Team Hector Rescue Robots [Dar12]

## 1.2 Problem description

It has already been shown that there are many different devices available - all starting at a price range above educational or leisure activities. Therefore, one of the key aspects of this thesis that needs to be evaluated will be the costs of robotic ground vehicles. Hence, we need to define a good robot, explore its hard- and software requirements while keeping the costs as low as possible and sustaining most important features.

This thesis follows a low cost approach for building an advanced UGV by examining existing solutions, current hard- and software and by creating a basic modular design. We will furthermore reproducibly implement the results with inexpensive components and evaluate the created robot platform called ***a** **Mo**bile **Se**nsor **Ro**bot* (aMoSeRo).

## 1.3 Structure

First, we examine the general software requirements of mobile robots and compare self-written solutions with robotic frameworks. After that we need to have a closer look at the advanced hardware demands and define functional modular components an UGV consists of. Finally, we will implement a low cost UGV.

# 2 Software

In this chapter we investigate different software solutions for UGVs. Defining the general requirements and relating them to our special robotic case is one of the first issues to solve. Next, we estimate the workload for implementation and development of a self-made UGV operating system and compare it to using different already existing robotic frameworks. The main metrics we are going to use are the efforts and expenses of necessary implementation time. Finally, we conclude our perceptions into a concrete decision for our low cost UGV.

## 2.1 Requirements

Beside usual software quality characteristics like efficiency and readability, robotic software has further requirements. Those e.g. are individuality of robot dependent code, distributed concurrent computations and complex software environments.

### 2.1.1 Software Reuse

Creating a mobile robot system from scratch includes a lot of programming effort. Besides choosing an efficiently performing programming language and developing a suitable environment, most of the created code usually tends to become robot dependent. Numerous robots have already been built in that way. Especially university robot prototypes, which are often only built a single time, come with a critical disadvantage: reduced reproducibility. To verify or reproduce research results, scientists practically need physical access to the created robot, as often there is no adequate simulation environment available and the robot construction and code is not publicly available. Therefore, external scientists frequently build their own robots and by that enter the circle of wheel reinvention [WG10].
An approach to solve this issue is the *Willow Garage PR2* which can be understood as a well defined robotic platform. By abstracting common tasks into reusable components with well defined interfaces and parameters, most solutions become independent of the current robot and can be reused in other projects. That remarkably reduces development time, cost and furthermore improves software quality. Research results created on a single *PR2* can be verified on every other *PR2* by simply installing corresponding components and running the created code. Because of hardware abstraction the same code can even be run on other robots that follow the same open principles.
Moreover, a solid platform allows the creation of reusable visualization, simulation and testing software for multiple robots and therefore allows heterogeneous robot setups. A fact most single purpose applications can not achieve as they tend to omit surrounding components.
Like for most cases, using existing software to the highest possible amount should be preferred for our project as it delivers more functionality and saves programming expenses. Nevertheless, hardware dependent code needs to be created and subsequently robot independent interfaces are inevitable.
In conclusion, the optimal software of a robot should abstract tasks into reusable components with well defined interfaces and parameters. By that, solutions become independent of the current robot and can be reused in later projects, significantly reducing long term development time.

### 2.1.2 Distributed computation

Since most mobile robots are not single computational systems, they are usually running on distributed hardware like integrated circuits, micro controllers and multiple processing units. Therefore, well defined interfaces between these parts are essential in order to replace single components or communicate across different programming languages and devices.
Furthermore, only fully autonomous robots do not ultimately require a network component or any other communication interface. In most cases UGVs are only the executive part of a more powerful computational system. Typically, at least one strong server provides CPU-intensive services like mapping, localization or planning as these computations require corresponding hardware specifications and a lot of power, which currently can't be provided by mobile devices.
Additionally, most robots do not include any sort of visual output device. Moving visualization data to a *Graphical User Interface* (GUI) via network communication may be an additional requirement.
In summary, all interfaces in an optimal software should be able to communicate over different bus systems like serial, *Universal Asynchronous Receiver/Transmitter* (UART), *Universal Serial Bus* (USB) or network. Additionally, well defined interfaces often reveal recording functionality and by that improve simulation and optimization.
Implementing a distributed computational network with concurrent running tasks requires advanced knowledge of all platforms that the software shall be running on. Moreover it is a task most robots require. Like it has been explained this can efficiently be solved by abstracting it to a robot independent solution. Therefore, most *Robot Application Frameworks* follow that principle in different ways. Some of the most important ones will be presented in the next section.

### 2.1.3 Robotic Software Environments

Next to the software running on an UGV, a lot of other applications are required. We now examine the three most important parts that are not easy to differentiate as they tend to merge into each other.

#### Visualization

Visualization is required for practical advanced human interaction and situation analysis. Hence, envisioning the robot according to its environment using cameras, maps and 3D point cloud data is an important task. Furthermore it is used as an intermediate step for higher programming concepts and non-hardware-aware software applications like behavior or human interaction. Interpretations of this abstraction are visual programming languages.
Another essential aspect is the visualization of the robots internal status. In more detail, displaying hardware statistics, interface data flows, transformation status publishing rates and capability graphs is important during development and for the decision making process of a robot and require further programming. Subsequently, visualization increases time efficiency during incorrect situations like appearing hardware errors or semantic issues.

#### Simulation

Usually, robot platforms are expensive and physical - which is why there are often multiple programmers working on a single robot. In consequence, not all of the programmers can test their software at the same time. To solve this, most developers make use of simulation environments as it significantly speeds up development and by that is helping to produce more stable software. A prerequisite to that is a simulation environment capable of emulating the real world and its corresponding physics. Consequently, simulation is especially helpful while programming collision avoidance and balancing experiments as well as it allows the preparation, structurization and investigation of recorded data.

#### Testing

Testable code improves software creation by every measure. The most common case is the question if code extensions or bug fixes influence other parts of the software. With automated tests this question can be answered faster and more correct than manually by a programmer. But possibilities of testing go beyond this feature, for example *Continuous Integration* (CI) and *Test Driven Development* (TDD) are advanced development concepts that support multiple programmers and aid to create stable and working code under the premise of correct usage.
Furthermore, the demand of being testable often positively influences code structure and, especially at large code bases, enables a higher degree of manageability. Because of that, most complex software uses automated testing in some way to guarantee functionality and extends its test portfolio with every fixed error.
On the other hand, testing can be time consuming and error-prone. Especially in small projects automated tests significantly slow down development progress when overused.
At the beginning, tests of our own software are optional, but should be able to be implemented afterwards - this is not the generally recommended way, but a pragmatic concession to the limited time.

### 2.1.4 Licenses

In case of using existing *Robotic Software Environments* written by other programmers, we need to mention different licenses.
There are three main types of licenses that need to be differentiated: first proprietary software, which does not make its source code openly accessible and often is connected to fees when being used. These closed source systems are mostly created by companies which sell a product and by that often produce less secure software than others by following the concept of security through obscurity. Because of that all additional features, maintenance and bugfixes need to be achieved by their internal programmers who need to be paid.
Next, open source software. Coming under different licenses, like the *Open Software License* [Ini14] (OSL) or multiple versions of the *GNU General Public License* (GPL), this software publishes each piece of source code available for everyone. By that, everyone who is capable of programming theoretically can review and improve the software in matters of security and functionality. It further has been shown that the code written by multiple programmers in open source projects is remarkably better and maintainable [Cov14]. Another essential point to mention is that open source licenses often come with some restrictions in code usage, like requiring the copy left principle or prohibition of military use. That aside, most of them allow free of charge commercial use.
A more specified subset of open source software is the so called free software, which truly frees the software from any restrictions of usage. The most popular examples for free software licenses are *Berkeley Software Distribution* (BSD) and the MIT License.

## 2.2 Robot Application Frameworks

A robot framework is a reusable set of libraries or classes for a robotic system. As already discussed, such frameworks enable developers to avoid a lot of unnecessary basic work. Unfortunately, frameworks tend to have a high level of complexity and consequently have steep learning curves until they reveal their full benefits.
Some of the most important and widely used frameworks are introduced in the following sections.

### 2.2.1 Microsoft Robotics Developer Studio *(MRDS)*

The MRDS is a windows based proprietary environment to control and simulate various robots. To allow high level programming languages like C# to be used in a robotic context, the MRDS among other things extends the *.NET-Framework* by adding a managed code library called *Concurrency and Coordination Runtime* (CCR), which manages inter-process messages and asynchronous operations [Mic12a]. Furthermore, *Microsoft* developed a software principle on top, named *Decentralized Software Service* (DSS), which provides a state-oriented service model that implements *Representational State Transfer* (REST) on system-level. This can be used for building high-performance scalable applications which run on a single node or across a network [Mic12b].
The MRDS offers some very useful features. First, development of web-based and windows-based GUIs become significantly facilitated. At the same time, programming has been simplified by the *Microsoft Visual Programming Language* (VPL), which allows easy access to a robots actuators and sensors, principally via drag and drop. Next, the MRDS makes it possible to add other services or libraries, like they are provided by *Open Source Computer Vision* (OpenCV) or of their own products like the *Microsoft Kinect*. Another point is 3D hardware acceleration during simulations. This can be remarkably faster than on open source operating systems. Finally, it supports a lot of common robots by design, like the iRobot *Roomba*, *iRobot Create*, different *Lego Mindstorms* robots and even *Segways*. As a result, the MRDS benefits from a relatively low learning curve.
However, the framework has not been very active over the last three years.

### 2.2.2 Cyberbotics - Webbots

Another powerful proprietary robot development environment that needs to be mentioned is *Webbots* which has been created by the spin-off company from the *Swiss Federal Institut* named *Laboratoire de Micro-Informatique* (LAMI) in 1998. After over 17 years of existence, today, it is capable of designing complex robotic setups in shared three-dimensional environments with one up to multiple robots of the same kind or even several different robots. The properties of each device like shape, color, mass and friction can be chosen by the user and even can have different locomotion schemes like wheeled robots, legged robots, or flying robots. A condition for using the Webbots *Integrated Development Environment* (IDE) is basic knowledge of *C*, *C++*, *Java*, *Python* or *Matlab* programming. However, the project is well documented and offers multiple libraries and APIs.
Because of its high focus on simulating robots in physically realistic worlds it depends on real units and as a consequence increases development speed and quality. Based on the *Model Program Simulate Transfer* approach [Mic04], the simulated robots later can be transferred on real, mostly commercial, devices. Not surprisingly, *Webbots* is very suitable for education [GLHF11] and therefore has already been used in over one thousand universities and research

centers worldwide [CYB14a].
Webbots runs on Windows, Linux and Mac OS X and comes in different versions and proprietary licenses (**Tab.1**). Some features like robot programming or the fast simulation mode are essential for productive robot development. Other features like the so called supervisor mode, where a single robot can control other robots or the simulation environment, are optional.

| **Webots feature** | **FREE** | **NAO** | **EDU** | **PRO** |
|---:|:-:|:-:|:-:|:-:|
| Price in CHF | 0 | n.a. | 320 | 3500 |
| Supervisor capability | no | no | no | yes |
| Physics plug-in programming | no | no | no | yes |
| Fast simulation mode | no | no | no | yes |
| Robot programming | no | limited | yes | yes |
| Transfer to real robots | no | limited | yes | yes |
| One year Premier Service included | no | yes | yes | yes |
| Robot and environment modeling | yes | no | yes | yes |
| Multi-platform: Windows, Mac & Linux | yes | yes | yes | yes |
| Floating & dongle licenses | N/A | yes | yes | yes |

**Tab. 1:** the Cyberbotics - Webbots versions and features overview [Cyb14b]

### 2.2.3 Player Project

The Player/Stage Project has been founded by Brian Gerkey, Richard Vaughan and Andrew Howard in 2000 at the University of Southern California at Los Angeles [Pla12]. It is a follow up of *Arena* and *ArenaServer* they had written before. The name originates from a famous William Shakespeare cite "All the world's a stage; and all the men and women merely players" from his play *As You Like It* [Sha04].
In this regard, the robots are the players, and the world coordinate system in simulation and the world is the stage. The *Stage Simulator* has been a 2D multiple robots simulation built on top of *FLTK*, a cross-platform *C++* GUI toolkit. Subsequently the *Stage Simulator* has been able to simulate hundreds of robots at the same time in a basic simulation environment. Later, the *Stage Simulator* got extended and replaced by another project, called *Gazebo*, which, among other things, supports three dimensions and furthermore provides distributed calculation capabilities. Henceforth, the environment is called *Player Project*.
The players in the project are the robots that need to implement a *Player Robot Abstraction Layer*. This layer can be applied to both, simulated robots and real devices. It provides support for various programming languages like *C*, *C++*, *Python* and *Ruby* and its socket interface is as unprescriptive as possible. *Player* which runs on *POSIX* compatible Systems like *Linux*, *MacOS*, *Solaris*, *BSD* and *Windows* and proclaims itself as one of the most used open source robot interfaces in research and education [CTHJMB05].
The *Player Project* is licensed under GPL and - like we introduced in *Licenses* - therefore is open source copy left required software.

### 2.2.4 Robot Operating System *(ROS)*

The *Robot Operating System* (ROS) is an open-source meta-operating system which provides essential features, namely hardware abstraction, low-level device control, implementation of environmental functionality, such as visualisation, simulation or testing and allows message-passing between concurrent running processes [O'K13]. Furthermore, it offers imple-

mentations of commonly used functionality in installable packages which even cover complex algorithms like *Simultaneous Localization and Mapping* (SLAM) and *Visual Object Recognition* (VOR). ROS moreover contains tools and libraries for obtaining, building, writing and running code across multiple heterogeneous computers [ROS14] and therefore includes language and platform independent tools [Ng10]. For example, ROS supports multiple client libraries, namely **roscpp** for *C++*, **rospy** for *Python*, **roslisp** for *Lisp* and many others. It is also possible to link application-related code and external libraries like *OpenCV* for computer vision or *Eigen3* for efficient linear algebra computation. Furthermore, ROS can successfully be wrapped around other frameworks like the *Player Project*.
ROS is mostly licensed free and has been developed as open source software under BSD Licence. Like discussed in *Licenses* on page 17 this offers a variety of advantages for a low cost robot.
Unsurprisingly, with the high complexity of ROS there comes one of the highest learning curves of all robotic frameworks. Besides, due to rapid changes of main characteristics during different major versions, nearly all books and most tutorials on the internet became unreliable which is often very confusing for a beginner. But, after the top of the curve, a lot of things are self explanatory and complex features can be implemented very fast.
Unfortunately, another point to mention and one of the main disadvantages of ROS is the dependency on the ROS host and its *Operating System* (OS). In case you do not develop on a *x86 32bit* system a lot of automations do not work and require patience to be solved. Especially, the support of packages on *armhf*, the *ARM* release repository, is not very usable, yet. Additionally, despite the importance of reactivity and low latency ROS is - like all other frameworks - no realtime OS.

### ROS Terminology

ROS is a message-based concurrent running heterogeneous peer-to-peer network application. Its structure can be imagined as a mostly undirected graph with an obligatory center process node, called **roscore**. Broadly speaking, this one master node tracks every other part of the robotic network, including running processes and their interfaces. The centralistic design consequently uses its advantages by offering global debugging possibilities and error logging. It further mediates direct connections between every graph node on request. This becomes very useful in cases like image processing, where running traffic over the central node would impact the global system by increasing network usage and processing power.
Still simplifying, other parts of the graph are organized name spaces, called **rosnodes**, which in turn are containing more **rosnodes** or process edges called dependently on their function as **rostopics** or **rosservices**. A **rosnode** in a ROS environment therefore can be a robot, a processing server for navigation or even a human interaction device, like a laptop. Usually they physically do not cross the border of a single computing system, but often a single system can run multiple name spaces. Also, **rosnodes** profit from zero copy shared memory handling between their topics by using the ROS **nodelet** manager and by that significantly reduce memory consumption. Every **rosnode** offers at least one **rostopic**, a multi-peer subscribable message provider, or a **rosservice**, a bidirectional unique connection between peers containing parameters.

### ROS Package structure

The meta-operating system character of ROS offers a wide collection of development related tools. One of the most important is the package management feature. Similar to other OSs

it encapsulates functionality into easily installable units. There are two stages of packaging. The first one is using the current OS packaging system like most linux derivates usually have. After adding the ROS repository to the *sources.list*, common packages like the ROS visualizer **rviz** become available, which are platform dependent and often contain precompiled parts. This is the reason why it can be comfortable but confusing and often does not lead to the most recent versions of the packages. This is important when switching ROS major versions (e.g. from ROS *Hydro* to ROS *Indigo*), like it happened during this thesis.
On the other hand, we can make use of the current package development repository to access the most up to date version. Therefore, every package comes with its multi-programming-language source and has to be compiled on, or at least cross compiled for, the platform it later runs on. ROS also has tools to handle inter-package dependencies (**rosdep**), updating heterogeneous code-managing repositories (**wstool**) and a higher level building tool (**catkin**). In order to work with those tools correctly, every package consists of at least two files. The *package.xml* file contains detailed information about the ROS package, like author, origin, license or meta-dependencies. Next, *CMakeLists.txt* declares inter process messages, services, actions and further adds **make**-dependent information like linking sources, install and testing routines.
Another equally important file type that needs to be mentioned are **launch files**. Written in *eXtensible Markup Language* (XML) they usually contain well defined node setups and allow more comfortable parameter handling than starting all nodes and topics each by a single command. Consequently, they should be preferred to launch functionality.

### ROS *Transformation*s (TFs)

One of the most important packages a ROS robot should implement is TF, because it enables the robot to keep track of multiple coordinate systems (frames) and their relations between each other over time. Following the *ROS Enhancement Proposal*s (REPs) especially **REP105** the most global frame should be the *world frame*. Every other frame derives from it in a tree structure and can be transformed back into world coordinates by using the same units of measurement defined in **REP103**.
Another important frame tree is the robot itself. Starting with a mobile *base_link* further attached elements called *links* like wheels or cameras have their own frame and are connected via relations, also called joints. Those joints can be static or dynamic. A sample configuration can be seen in **Fig.16** and **Fig.17**.
To define a robot, ROS offers a special XML description file using the *Unified Robot Description Format* (URDF) which is further improved by special markups and an additional interpreter called *XML Macros* (XACRO). In ROS, all not time-related relations can be defined in a single file and can be published periodically by the **robot_state_publisher** for example for simulation purposes. In advanced setups, publishing the robots joint states and especially the relation of the *base_link* is a complex task. Therefore it gets divided into separate processes like navigation, mapping or the hardware controllers.

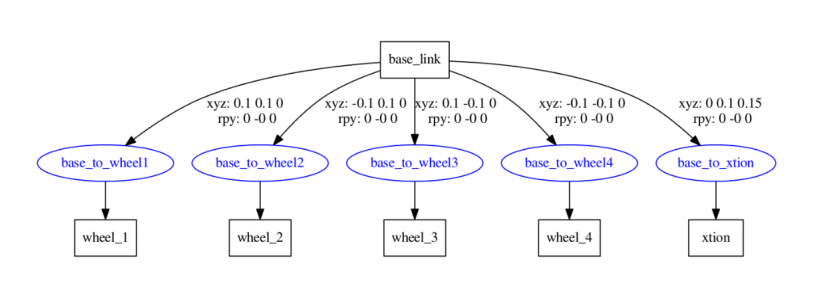


**Fig. 16:** ROS a simple robot TF tree, illustrating a mobile base with four wheels and a *Asus Xtion Pro* depth sensor

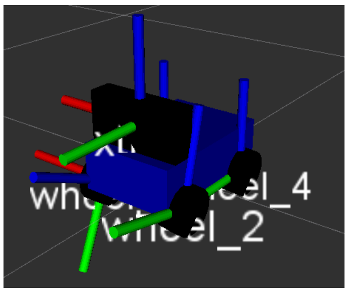


**Fig. 17:** ROS **rviz** visualization of the example TF tree

## ROS History

In 2007, the first robot running a version of ROS was *STanford Artificial Intelligence Robot* (STAIR) which was developed by *Stanford Artificial Intelligence Laboratory* (SAIL). During that time ROS was called *switchyard* [Ng07] but already followed its main principles like inter-process communication, concurrency and heterogeneous environments. After that, *Willow Garage* primarily developed ROS until February 2013. At this time ROS reached the critical-mass, every open source project needs to survive without being mainly driven by external funding. Since then the stewardship of ROS has been moved to the OSRF [OSR13] and subsequently left *Willow Garage*.
Major versions of ROS are called distributions and are named using adjectives that start with with successive letters of the alphabet. Starting with *box turtle*, *C Turtle*, *diamondback*, *electric*, *fuerte*, *groovy*, *hydro* and finally *Indigo*, which is available since May 2014.

### 2.2.5 Conclusion and consequences

The high demands of good robotic software environments and the complexity of base algorithms are outstripping the possibilities of a single programmer. Moreover wheel reinvention cuts the possibilities of efficient extensibility and prevents reconciliation or maintainability. Furthermore, most robots are facing abstractable issues like navigation, collision detection or efficient sensor data computation which is why code re-usage should be a prime directive.
Hence, this chapter briefly introduced the most important *Robot Application Frameworks* and *Robotic Software Environments*.
Unfortunately most proprietary and therefore cost intensive frameworks like the MRDS and *Cyberbotics - Webbots* are less suitable for a low cost robot. That aside their lack of extensibility and community activity are critical when facing programming issues and errors.
In our special case of creating a low cost robotic platform, we decide us to use an open source project with enough activity to persist the future: ROS.

# 3 Hardware

This chapter focuses on hardware related challenges surrounding robotic platforms.
First, we have a look at the general requirements and define low cost, conditions of software and physical requirements. Next, we introduce a modular design concept by differentiating between *Sensors*, *Accumulators*, *Processors* and *Actuators*. Finally, we discuss several options for each of their components, before specifying a complete implementation of an UGV in the next chapter.

## 3.1 Requirements

The hardware properties of a robot are defined by requirements like mechanical load or environmental factors and external parameters, namely money, availability of parts and ongoing technical progress.

### 3.1.1 Low cost

As already mentioned, UGVs like they are found in industry, education or *Do it yourself* (DIY) communities are currently not affordable for average technique enthusiasts, teachers in schools or sometimes even universities. The concept of low cost robots tries to solve that issue.

**What is low cost in a robotic context?**
The traditional interpretation of low cost is minimizing the expenses while keeping most important features. In borders of mostly expensive robotics this term needs to follow the same differentiation as between *cheap*, which means coming with a significantly reduced price and quality, and *keen*, considered as maintaining a certain amount of quality at a reduced total cost. For example, the 50 000 USD [WSJ14] UBR1 [Rob14c] is a low cost version 250 000 USD up to 400 000 USD *PR2* [WG14e] of *Willow Garage*, but still is far away from the term *cheap*. Another example and at the same time another robot Melonee Wise worked on is the *TurtleBot*, which was constructed with the attempt to be the lowest cost version of a ROS robot at time of creation.

**How to achieve low cost?**
There is no general solution to this problem. But an approach to solve the issue in the robotic context is to replace expensive single purpose solutions produced by companies in low quantities with mass produced products that get customized to suit the application.
A demonstration of this *positive misuse* are the first versions of the *TurtleBot*. Instead of constructing the robot with expensive 3D Laser Scanners they replaced it by a *Microsoft Kinect* originating from the gaming industry. Furthermore, it used a *iRobot Roomba* and later a *iRobot Create* as a low cost mobile base as constructing a custom moveable footprint would have been way more expensive. Also, the mass produced product came at a lower cost and unharmed warranty. An important side-effect of these replaceable parts is the independence of unique cost intensive and sometimes, due to customs regulations, not easily accessible parts. By that, the power to choose a cheap replacement at any time reduces overall expenses and total risk.

As a consequence, our UGV should be easy to build and reproduce, affordable for education and able to run ROS with some kind of 3D measuring device. It further should consist of easily achievable or replaceable parts.
In conclusion, these properties lead to a modular design concept with communication interfaces between the inexpensive components. Also a certain degree of flexibility is required to maintain extensibility and independence of expensive parts.

### 3.1.2 Software conditions

As we introduced in the *Software* chapter, applications in robotics need to solve a lot of computational intensive tasks. While some of them can be outsourced to an externally powered device like a laptop or a server, others essentially can be calculated on the UGV.
Examples for that are collecting sensor data, receiving and executing commands or streaming data. Balancing these is a challenging task, because on concurrent executing systems all processes can influence each other. Especially when computational power gets cut down to the limits in order to save energy. As most libraries, frameworks or software environments do, ROS requires additional resources when being compared to a single purpose application.
In conclusion providing enough computational power while using reasonable amounts of energy is an important task to solve.

### 3.1.3 Physical properties

Physical dimensions and requirements result from a tradeoff between costs and size, whereas smaller UGVs tend to be more expensive and complex. On the other hand, an upper bound among others is set by being manageable in terms of transport and storage.
The target UGV for this thesis is a four wheel or two tracks driven ground robot with physical dimensions below $150mm * 300mm * 300mm$ (height, width, length). The drive power should be accordingly with an effective force of more than $100Ncm$ for moving or holding torque in case of stronger slope. Additionally, tracks are the preferred primary propulsion system as they have better grip properties and only require simple motor control. Another *nice to have* would be the capability of spot-turning, which would allow operating on small areas and facilitates 3D scans of rooms without moving further than required. Another optional point if the robot is going to be used outside of buildings or around kids is a splash-proof case that would increase the robots life. Furthermore, modular extensibility would increase the usability of the robot significantly.

## 3.2 Modular design

Like the graph in **Fig. 18** shows, we divide the functionality of UGVs into four main modules: First, Sensors are the parts the robot requires to sense the outside world, next *Accumulators* serving and saving power, followed by *Processors* the units are processing information gathered by *Sensors* and finally, *Actuators* which provide physical movement. These areas in turn get separated into further sections which we discuss one by one on the next pages.

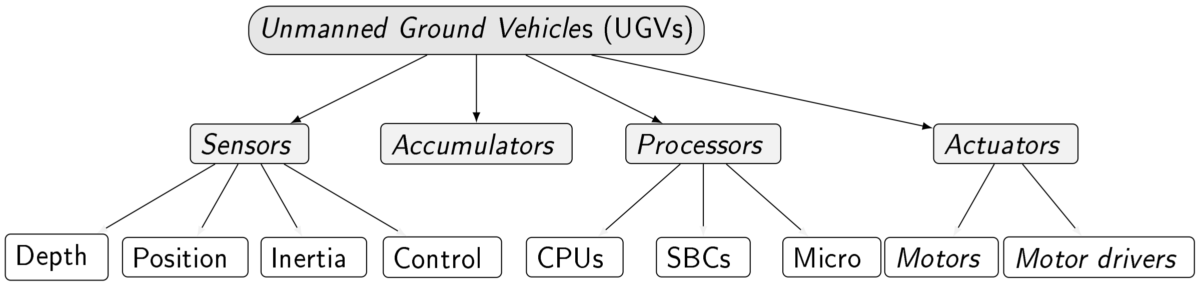


**Fig. 18:** Functional modules of UGVs

### 3.2.1 Sensors

A sensor is a device which measures physical quantities and converts them into electronic signals. A robot reads these differently shaped signals by using according kinds of communication interfaces. Often connections are established through serial sockets like USB, *FireWire* (IEEE 1394 High Speed Serial Bus) or an *rs232* port. Some more hardware-related systems like SBCs support programmable *General-Purpose Input/Outputs* (GPIOs) or *Inter-Integrated Circuits* (I2Cs) and therefore are able to emit, receive or measure current power or communicate to addressable integrated circuits.

#### Depth Sensors

Measuring the surrounding environment of a robot is an essential task for further applications. There are various ways available with very different accuracies and prices. Now, we delve into the most important of them, starting with the previously mentioned hardware-related solutions.

**Ultrasonic Range Scanner** Equally to light, sound needs a certain time to travel. It is possible to determine the distance by sending out a sonic signal and listening to its reflected echo. The time of travel is estimated by measuring the difference in time between the pulse being initiated and the echo being reflected. Ultrasonic sound by definition are oscillating sound waves over the upper bound of the human hearing range. This limit varies but usual starts over $20000Hz$ for young adults. A common example in nature where these waves are used for navigation are bats.

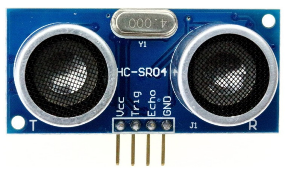


**Fig. 19:** the HC-SR04 [Aim14]

During this thesis, we experimented with a *HC-SR04* as shown in **Fig.19**. The sensor, which comes at a cost of $2EUR$ has four pins. One for applying operating voltage, another for providing ground, a third one to trigger a measurement and a fourth one that gets activated,

when an echo has been received. The outcome of various tests are a very low accuracy for distances above one meter and a high error incidence in cases on inclined surfaces.
As a consequence, these depth sensors are only suitable for a short range obstacle detection and because of their low price could be built in multiple times inside a simple UGV.

**Double Webcam Approach** Another way to estimate distances between a robot and its surroundings is the approach most animals and humans are following: stereoscopic vision. The utilization of two separate eyes and fusioning optical images into a three-dimensional picture are successfully imitated by a robot by replacing organic eyes with two high resolution web cameras. ROS already supports visual algorithms which, after non-optional calibration, provide proper visual odometry.
Based on findings by Martinez Fernandez [MF13] it can be argued whether the achieved results of these algorithms are accurate enough, especially in comparison to other methods.

**Fig. 20:** the Hokuyo's URG-04LX-UG01 [Ha14]

**Laser-Scanner** Laser is a special form of light. *Lasers-Scanners* use the steering of laser beams to emit a special pulse to every point of sight while measuring the time the beam needs to get reflected by the surrounding area. This principle is called time of flight. It furthermore facilitates usage of the *Doppler Effect* to determine whether an object currently moves forward or backward in relation to the scanner. On the other hand, most laser-scanners or laser-rangefinders can be jammed during normal operation - for example by smoke, shiny, mirroring or transparent objects, which reduces their reliability.
One of the inexpensive variants are laser-distance-measuring-devices which can be found at home improvement stores to measure single indoor distances. If these are suitable for a low cost robot has not been determined during this thesis, but estimating their effort to be used effectively in ROS, they at least require a lot of knowledge or time to get set up. Other devices with the ability to scan multiple dimensions in a short period of time are significantly more expensive and are mostly used by the surveying industry. These scanners often contain periodically moving mechanical mirrors which increase power consumption, error susceptibility and weight [Dey08].
Eligible examples with acceptable expenses and general technical conditions for a low cost robot are the devices manufactured by *Hokuyo* with prices ranging between 1140 and 5875 USD like shown in **Tab.2**.

| Name | scanning interval | detectable range | angular resolution | operating voltage | weight | Price USD |
|---|---|---|---|---|---|---|
| URG-04LX-UG01 | 100 msec | 240° 20mm to 5600mm | 0.36° | 5V | 160g | 1140.00 |
| UXM-30LAH-EWA | 50 msec | 190° up to 80m | 0.125° | 10-30V | 800g | 5875.00 |

**Tab. 2:** Hokuyo's Laser-Scanner price range bounds

Another option to save costs is disassembling vacuum cleaner robots, which start at prices about 400$USD$ and use some cheaper versions of *Laser-Scanners*. Depending on model and level of documentation, the risk of damaging the laser or the possibility of being used at all can not be estimated at this point.

**_Light radar_ (Lidar)** *Light radar* (Lidar) is a portmanteau of the words Laser and *RAdio Detection And Ranging* (RADAR). Invented in the early 1960s, it first was used by the *National Oceanic and Atmospheric Administration* (NOAA) to measure clouds [On14] and later played an important role during the creation of maps of the moon by Apollo 15 in 1971. It still is used in space docking maneuvers and extra-terrestrial landings.
On first sight, the physical properties are similar to *Laser-Scanner*s, but instead of only measuring time of the reflected light, Lidar further emits ultraviolet and infrared light waves to analyze the thrown back wave lengths. Because of various forms of diffuse reflection caused by scattering, Lidar can even reach resolutions at the molecular level.
In the context of mobile robots Lidar is often used as a more exact version of *Laser-Scanner*. The prices of 3D systems are with 70000$USD$ comparably high. Consequently, only well founded projects like the *TU Darmstadt - Team Hector* or the *Google Car* project can afford Lidars.

**Microsoft Kinect** The *Microsoft Microsoft Kinect* is a horizontally designed sensor bar. The main task of the device is turning the humans body into a gaming controller by motion capturing the bodies of multiple players at once. As the work of Thomas Kühn [Kue11] examined the internal design, we know that the *Microsoft Kinect* is uniting a multi-array microphone with a RGB- and a special range-camera developed by *PrimeSense*. By using on board algorithms, it provides 640x480 pixels of $2^{11}$ possible depth values at up to five meters of range. Comparing its price of currently 100$EUR$ to other solutions, especially like *Laser-Scanner*, the *Microsoft Kinect* is a considerable low cost solution for depth sensing.
At release, all software around the devices had been proprietary and for the *Microsoft Xbox* only. Soon, this changed because of high interest of the open source community and by Hector Martin [Ada10]. After the so called *libfreenect*, *Microsoft* released their *Kinect .NET Development Framework*, which now enables all platforms to develop custom applications.
In comparison to the next device we are going to examine, the *Microsoft Kinect* sometimes suffers from hardware issues when used in multi-device-environments [iPi13] e.g. two robots looking at the same area at once will cause errors in data.

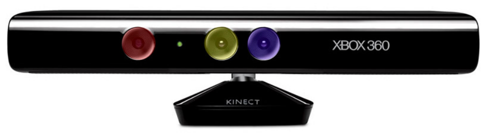


**Fig. 21:** *Microsoft Kinect* - with one infrared emitter (red), a rgb camera (yellow) and depth camera (blue) ©*Microsoft*

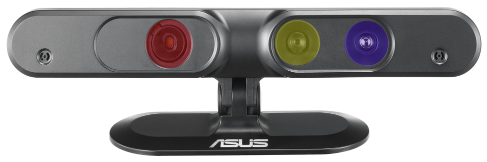


**Fig. 22:** *Asus Xtion Live* - with one infrared emitter (red), a rgb camera (yellow) and depth camera (blue) ©Asus

**Asus Xtion Pro** The *Asus Xtion Pro* is a PC-clone of the *Microsoft Microsoft Kinect*, developed by *Asus*. It uses the same core 3D sensing solution from *PrimeSense*. Unlike the *Microsoft Kinect*, the *Asus Xtion Pro* was developed primarily for browsing multimedia content, accessing web sites and social networks. Besides the smaller physical size of the *Asus*

*Xtion Pro*, it furthermore does not need an additional power supply. By that, the *Asus Xtion Pro* does not to require more than the USB version two maximum current power, which is $500mA$.
The library which accesses the *Asus Xtion Pro*'s data *OpenNi2* was not available for *ARM* at the point of writing. After some additions to the source code and custom compiling on our own hardware, we managed the package to work with ROS and especially *catkin* both on *ARMv6* and *ARMv7*.
Furthermore, the *Asus Xtion Pro* is more compact than the *Microsoft Kinect* and weights much less ($0.5lb$ against $3.0lb$). On the other hand, because being less popular the smaller community had issues e.g. with handling all changes by USB 3.0 drivers. Although, it provides better RGB quality, but overall with $150EUR$ costs more than a *Microsoft Kinect*.

**Leap Motion** As partner of *ASUS* [Eng13] the *Leap Motion* uses the same emitted and reflected infrared light for tracking parts of the human body like the *Asus Xtion Pro*. Available since July 2013 [Tec13], the *Leap Motion* with about $90EUR$ is an inexpensive, but limited input device, shown in **Fig.23**, which is optimized for tracking fingers and hands as illustrated in **Fig.24**. The main features include the tracking of two simultaneous hands with gesture recognition for all ten fingers. For distances between $10cm$ and $1m$ at daylight the device works reliably.
During this thesis, we have tested the existing ROS driver, which currently only supports one hand and was not able to provide 3D PointCloud data. In brief, the *Leap Motion* unfortunately is inappropriate for our project as their only use could be unreliable robot control by hand gestures.

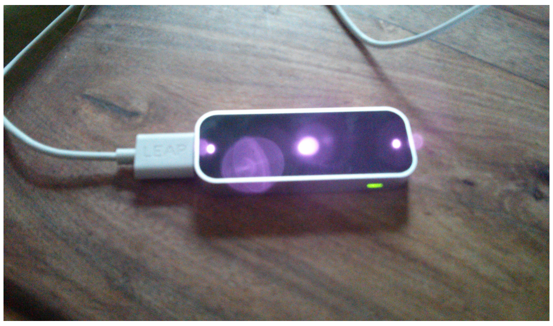


**Fig. 23:** LeapMotion device emits infrared light, which can be seen with a non-filtered camera

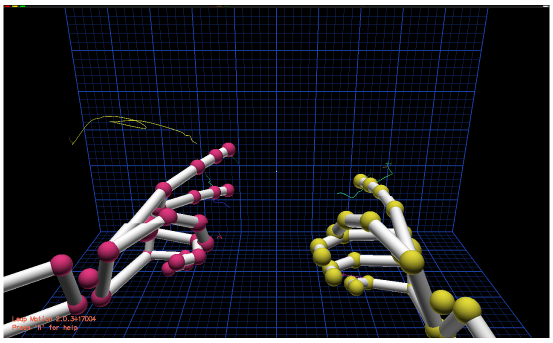


**Fig. 24:** LeapMotion Visualization API allows tracking of two hands and advanced gesture recognition

## Position

Particularly, *Depth Sensors* appear well-suited for positioning. Unfortunately, their accuracy tends to be too imprecise. Consequently, most robots are using at least one additional source and combine both, according to their relative quality among others by using linear quadratic estimation filtering, originally introduced by *Rudolf E. Kálmán* [Kal60].
To improve the data generated by *Depth Sensors*, it is possible to measure the position of wheels and how much they have turned by potentiometers. The so called hardware odometry itself requires advanced knowledge of integrated circuits and subsequently would require a lot of time to build. Obtained odometry data is one of the preferred sources among cheaper mobile robots like the *iRobot Create*. With *Stepper Motors*, which we investigate in 3.2.4, the

movement can be very well estimated by the steps they were initiated to run and probably would not require additional sensors.
Another well-suited sensor to determine positions is the *Global Positioning System* (GPS). It calculates the position by triangulating signals from satellites. These satellites have known positions and emit timestamp-data. When these arrive at a local antenna they can be triangulated according to their time of travel, and therefore at least allows an accuracy within a two-dimensional coordinate-system when used outside of buildings.
Another sensor type we used during this thesis was a barometer, which measures barometric pressure with high accuracy when used inside closed rooms. Nevertheless, a barometer highly depends on constant barometric circumstances and should not be used outside of buildings, where single air movement like winds can distort results significantly.

### Inertia

However, every UGV experiences inertia. By definition, Inertia is the physical resistance of an object to any change in state of motion. This includes changes to its speed and its direction. The active physical force can be inexpensively measured in several dimensions since the advancing smartphone-technology reduced the costs significantly.

#### *Inertial Measurement Unit* (IMU)

There are three main types of inertial position sensors between which a distinction needs to be made. First, the accelerometer, which measures proper acceleration (without effects of gravity) containing up to three axis called *Degrees of Freedom* (DOF). Next, the magnetometer is a sensor to measure the direction of the earth's magnetic field. With its help, it is possible to determine the direction of the magnetic north pole of the earth even while being upside down, a task where a usual compass would fail. On the one hand, their accuracy is very high, which is why they also can be used as depth ferrous metal detectors. On the other hand, they can be easily influenced by other fields, like the ones computers or humans do emit.
Finally, the gyroscope - a device to measure orientation based on angular momentum, which especially can be used when magnetometers do not work, e.g. in space, or when they are not exact enough.
An IMU combines these three types of physical force sensors and converts them into electronic signals. Because of the high frequency and the advanced amount of data inter-part protocols like I2C or *Serial Peripheral Interface* (SPI) or higher forms of communications are used.
We usually categorize IMUs by summing up their DOF. Because of our three dimensional world and so far three different sensors the maximum number is nine. Sometimes producers of IMUs add further position sensors like a Barometer or GPS and keep on adding those with one DOF each.
We experimented with several IMUs which are illustrated in **Tab.3** with an asterisk.

| **Name** | **Total DOF** | **Ac-celome-ter** | **Gyro** | **Magne-tome-ter** | **Com-muni-cation** | **ROS driver** | **Price EUR** |
|---|---|---|---|---|---|---|---|
| WiiMote* | 3 | 3 | - | - | Blue-tooth | yes | 42 |
| WiiMote+* | 6 | 3 | 3 | - | Blue-tooth | yes | 60 |
| LSM9DS0* | 9 | 3 | 3 | 3 | *Arduino* I2C, SPI | no | 18 |
| 9DRazor | 9 | 3 | 3 | 3 | onboard *AT-mega328* serial | yes | 100+ |
| more advanced solutions | 9+ | 3 | 3 | 3 | Serial | n.a. | 120+ |

**Tab. 3:** *Inertial Measurement Unit* (IMU) Overview

Another essential point to mention ts the different operating voltages between the LSM9DS0 ($3.3V$) and other comparable devices which are often run on $5V$. Connecting a *LSM9DS0* to a $5V$ circuit therefore requires a special bidirectional voltage level converter, which costs about $2EUR$ when bought instead of being assembled by parts, but significantly increases the complexity of the circuit. Equally important to be taken into account are different measurement scales and accuracies while transferring data into a ROS /**imu** topic especially because they need to suit the REPs suggested units of measurement.

## Control

In case a robot needs to be controlled directly by a human, a hardware sensor or interface is required to translate the human input into robot movement commands. Instead of developing our their controller, many robots use game controllers like entertainment consoles use.
This fits into the low cost concept and comes with an intuitive or well-trained feeling for most people. During that thesis we connected several console controllers to ROS as shown in **Tab.4**. As most of the common controllers already have functional drivers for *Linux* and therefore can be translated into ROS joystick or short /**joy** topics by code, most of the programming effort was creating a reasonable controlling concept for moving a UGV.

| **Name** | **Axis** | **Buttons** | **Connection** | **Feedback** | **Gyroscope** | **Price EUR** |
|---|---|---|---|---|---|---|
| Playstation 3 Controller | 2+2 | 17 | USB | Dual-Shock | no | 46 |
| Xbox Controller | 2+2 | 17 | USB wireless | Force-Feedback | no | 42 |
| WiiMote | 3 + 2 | 12 + 2 | Bluetooth | Sound Vibration | no | 42 |
| WiiMote + | 3 + 2 | 12 + 2 | Bluetooth | Sound Vibration | yes | 60 |
| LeapMotion | 3 | 0 | USB | none | computable | 90 |

**Tab. 4:** Teleoperation controller Overview

The basic principle behind all controllers is to generate valid *geometry/Twist* messages the robot can execute. By that, every controller is using the same recordable interface while most ROS algorithms still can be applied and react accordingly.

### 3.2.2 Accumulators

The ideal battery for our project is long lasting, light weight, fast rechargeable or at least easily replaceable and should be as cheap as possible. Unfortunately the battery research somehow still is the bottleneck of every mobile device and the fulfillment of the combination above is currently far from possible. So we need to carefully face our demands with the fact that with higher capacity usually comes higher weight - starting a vicious circle of higher motor torque requirements, more power consumption and higher battery capacity again.
If we look at comparable mobile robots, the usual battery service life is slightly above a single hour for the *TurtleBot* and below four to five hours for the *iRobot PackBot*. Some robots like the *PR2* further support to run while charging. We want to have battery lifetime at least within similar range.
Therefore we considered using multiple battery technologies for both, our $12V$ (up to $2A$) and $5V$ (up to $1.5A$) circuits. Some specifications are illustrated in **Tab.5**. For us, a smart phone charger pack with two different voltage exits is optimal.

| Name | Voltage in V | Capacity mA | esti-mated recharge time | Price EUR |
|---|---|---|---|---|
| AAA | 1.5 | 2000 | 10h (200mA/h) | 2 |
| 9V Block | 9 | 800 | 4h (200mA/h) | 3 |
| LiIon | 3.4 | 2000 | >3h | 10 |
| LiIon | 5.8 | 2000 | >3h | 10 |
| Battery Pack Smart Phone Charger | 5 and 12 | 15000 | 12h | 45 |

**Tab. 5:** Battery technologies

Understanding battery behavior in robotic projects requires deeper knowledge of electronics. Specifically, LiIon accumulators can be very dangerous as they heat up until burning or exploding if handled wrongly. Series and parallel circuits of batteries should be encapsulated as far as possible and reviewed by electronic experts. Moreover, power supplies for charging should be used according to their manuals to avoid harm.

### 3.2.3 Processors

The heart of every robot is at least one central unit to process its data. Requirements for processing differ according to the task the unit accomplishes. This especially is important when talking about time critical controls as we examine in the next chapter, *Actuators*, where *Microcontrollers* are essential.

#### Central Processing Unit*s (CPUs)*

For the CPU there is a variety of possible solutions available. Since the technological progress has been advancing processors due to the demands of an increasingly mobile future, we carefully need to decide from which our mobile robot would benefit the most.
The range of available devices reaches from integrated circuits, programmable micro controllers through single purpose computers, *System on a Chi*ps (SoCs) or *Single Board Computer*s (SBCs).
For us, the most distinguishable characteristic property for modern processors is the instruction set it is using. There are three main branches we need to have a closer look at:

#### x86 (32 bit) instruction set processors

*x86* is a family of instruction sets used since the early days of 16 bit architectures. In 1985, the *i386* was the first processor that extended the length of the instructions from *16* to *32 bit*. Since then, many manufactures implemented the same instructions on their units, which is why the *x86* family is still built into the majority of computers, besides the mobile segment like smartphones or tablets. The lion's share of existing software including drivers and frameworks has been built on and for this type of architecture.
Hence, both ROS *Hydro* and ROS *Indigo*, currently are optimized for *x86 (32 bit)* machines. On account of its wide spread, most packages, libraries and even drivers of the peripheral devices are available for *x86 (32 bit)* computers.

But as they are not produced for mobility from sketch, most *x86* processors use more power than other architectures we look at in the following sections. In addition to that they are limited to a specific amount of address space which reduces their maximum RAM and often sizes them down to single core CPUs restricting the multi-threading possibilities.
Unfortunately, also small and cheap SoC solutions with *x86 (32 bit)* are only available to a limited extent. Furthermore, this influences the limited amount on existing SBCs.
In 2003, *AMD* introduced the first *x86 (64 bit)* CPU which fixed incompatibility issues with the ongoing multi-core development.

### x86 (64 bit) / x64 (64 bit) instruction set processors

With the doubling of the maximum instruction length, *AMD* first reacted to the demand of multi-core architectures and bigger amounts of maximum memory used in a single machine by creating the so called *x64 (64 bit)* instruction set. Other manufacturers like *Intel* soon followed. Since then, the nearly equivalent *x86 (64 bit)* is the ideal choice for high performance tasks while usually keeping backwards compatibility.
Unfortunately, mobility and therefore overall power consumption, especially when compared to other instruction set devices, did not improve enough. That is why SoCs using *64 bit* are still relatively unusual. As a consequence, they are currently not qualified as a mobile robot processing unit.

### (32 bit/64 bit) instruction set processors

*ARM* instruction set processors are obstructed in smartphones and tablets which have been getting down-market products in the last few years. They are an advancement of the *reduced instruction set computing* (RISC), which in turn has been an improvement of the *complex instruction set computers* (CISC). As a result, they use significantly fewer transistors for equal commands. This reduces production costs, heat and power consumption and consequently forms them into eminently suitable devices for battery driven projects.
The *ARM* architecture has been introduced in 1985. In October 2011, *ARM* announced 64 bit versions of their architecture. Today these can be found in many mobile devices, starting with the *Apple iPhone 5S*.
They therefore are an appropriate choice for mobile robot processors.

## Single Board Computer*s (SBCs)*

Since connectors between components were a frequent problem for the reliability of a system, the solution was hard wiring them together into a single integrated circuit [Ros99][pp.50-51]. Following this approach, the first SBC called *MMD-1* was built by *E&L Instruments* in 1976. SBCs mostly have been used as increased density integrated circuits and were applied as backplane systems.
Starting with the *One Laptop Per Child Program* [one14] (OLPC), the development turned SBC into cheap price devices, unleashing their educational value. We considered using several SBCs illustrated in **Tab.6**.

| | RaspberryPi Model B | BeagleBone Black | CubieTruck | Banana Pi | UDOO | ODROID-U3 |
|---|---|---|---|---|---|---|
| **SoC** | Broadcom BCM2835 | AM3358/9 | AllWinner A20 SoC | AllWinner A20 SoC | Freescale i.MX 6 | SAMSUNG Exynos4412 Prime |
| **CPU** | 700 MHz ARM1176JZF-score | Cortex-A8 @ 1 GHz | *ARM* Cortex-A7 @ 1 GHz | *ARM* Cortex-A7 @ 1 GHz | *ARM* Cortex-A9 CPU Dual/Quad core @ 1 GHz | *ARM* Cortex-A9 Quad Core 1,7 Ghz |
| **GPU** | Broadcom VideoCore IV | PowerVR SGX530 | Mali-400MP2 GPU | Mali-400MP2 GPU | Integrated | Mali-400 |
| **Memory** | 512 MB | 512 MB | 2048 MB | 1024 MB | 1024MB | 2048 MB |
| **USB Ports** | 2 | 1 +1mini | 2+1mini | 2 + 1mini | 2 + 1mini | 3 |
| **Network** | Ethernet | Ethernet | Ethernet / Wlan / Bluetooth | Ethernet | Ethernet | Ethernet |
| **Low-level peripherals** | 8 $\times$ GPIO, UART, I$^2$C bus, SPI bus with two chip selects, I$^2$S audio +3.3 V, +5 V, ground | 4xUART, 8x PWM, LCD, GPMC, MMC1, 2x SPI, 2x I$^2$C, A/D Converter, 2x CAN bus, 4 Timers | 54 extended pins including I$^2$C, SPI | Extensible 26-pin headers | 76 fully available GPIO Arduino-compatible R3 1.0 pinout HDMI and LVDS + Touch (I2C signals) | I$^2$C, UART, GPIO |
| **Power ratings** | 700 mA | 210–460 mA | $<$ 500 mA | 240 mA | $<$ 2000 mA | $<$ 2000 mA |
| **Size** | 85.60 mm $\times$ 56 mm | 86.40 mm $\times$ 53.3 mm | 110 mm x 85 mm | 92mm X 60 mm | ca.100mm x 80mm | 83x48x20 mm |
| **Weight** | 45 g | 39.68 g | 290 g | 48 g | n.a. | 50 g |
| **Price EUR** | 40 | 52 | 89 | 45 | 99 | 70 |

**Tab. 6**: Single Board Computer Overview

**Raspberry Pi** The Raspberry Pi has been developed by the Raspberry Pi Foundation in 2011. The main intention behind the project is to support the computer science education in schools [RPi14a]. Since its release, more than 2 million devices were sold [RPi14b] by licensed manufactures, mostly to technique enthusiasts around the world. Because of being a

full-fledged computer while offering hardware interfaces like GPIOs and I2C it offers higher user-friendliness than the cheaper Arduino. The technical features are as shown in *Single Board Computer Overview* but as the first of its kind, its impact revolutionized the handling of most computer and micro controller based DIY projects.
During this thesis the *Raspberry Pi* suffered from its low performing CPU and was not able to process the 3D PointCloud data from the *Asus Xtion Pro* to ROS with more than $0.1Hz$. Furthermore, the power consumption was relatively high. Moreover, it showed instabilities during power fluctuations caused by powered USB devices.
The newly released version and nearly identically constructed successor model $B+$ promises better behavior but because of its unimproved CPU still seems unsuitable for a UGV.

**CubieTruck** As the successor of *CubieBoard 1* and *CubieBoard 2* the *CubieBoard 3* or its more popular name *CubieTruck* seemed to be a good replacement for the *Raspberry Pi*.
With the *A20* SoC it offers a variety of properties the *Raspberry Pi* would have needed. Especially, more *Random Access Memory* (RAM) and onboard flash memory were promising. While overall using less energy and the just slightly increased physical dimensions, we ran multiple tests and achieved all requirements like *Asus Xtion Pro* driver compatibility and running ROS.
On the other hand, the *CubieBoard* Community is significantly smaller than the Raspberry Pi's, which results in incomplete and sometimes even confusing documentation. This often led to unconventional workarounds to fulfil simple tasks.

**Other SBCs** During this thesis multiple SBCs have been released and their progress is steady. The *UDOO* and the *ODROID U3* are capable of running ROS while offering better hardware specifications. Especially the *UDOO* has the interesting feature of including a separate micro controller on board, eliminating an issue we are going to examine in Pulse-width Modulation *(PWM)* while being fully compatible to shields like we in brief discuss in *Motor drivers*. Considering their count of sold units will be smaller by a noteworthy margin, it can be concluded that their community and documentation status will be facing the same issues and might be slightly worse.
For those reasons, the *CubieTruck* was the best tradeoff between processing power and community size.

### Microcontrollers

When it comes to hardware control, timing becomes extremely relevant. One solution for this issue are single purpose circuits. Correctly constructed setups are able to process information very fast. On the other hand, building them requires advanced knowledge in electronics. Furthermore, expanding once constructed circuits is an nearly unmanageable task.
Another way to solve this issue are programmable computers, like we examined in previous sections. Especially, hardware related solutions which provide UART and I2C seem suitable, but as normal CPUs tend to run multiple concurrent tasks at the same time, interference between time critical tasks and computation intensive tasks can occur.
There is something in between the previously mentioned solutions: using dedicated micro controllers with well defined clock cycles.
A perfect example for this kind of hardware is the *Arduino* single board programmable micro controller family.

### Arduino

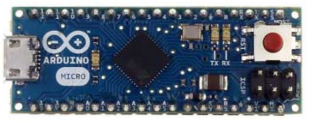


**Fig. 25:** the Arduino Micro [Ard14]

By merging usually an *Atmel AVR* microcontroller with programmable input and output pins, including digital pins and Pulse-width Modulation *(PWM)*, the *Arduino* open hardware solutions are often used in small electronic projects by technique enthusiasts. Because being comparably easy to program with a simplified *C* language, it become a recent option for programming education or art projects.
The Arduino hardware platform is creative commons licensed, and since the beginning in 2005 over 700 thousand official boards were sold and about the same amount of unofficial clones [Uni13].
There is a variety of different Arduino boards available, we tested a version of the currently most recent *Arduino Micro*, which is shown in **Fig. 25** and beside a decent size offers an integrated USB serial controller. By that, no additional serial programmer to write the programs onto the flash memory is needed, like other Arduino*s* would require. Furthermore communication to SBC can be established easily by USB.

### 3.2.4 Actuators

Actuators are devices to move or to control the movement of a robot. In our case, the moving parts come down to a mobile base, which in ROS is described by two terminologies. First, following **REP105** the preferred term is *base_link*. On the other hand the *PR2* example implementation and therefore very commonly used one is *base_footprint*. Both of them describe a propulsion system that is able to move the robot under certain perimeters.
We have already familiarized ourselves with some very popular preassembled mobile bases, like the *iRobot Create* or the *iClebo Kobuki* used by the *Willow Garage TurtleBot 2*. Specifically, *Kobuki* would be an ideal base for indoor environments - but with its price of around $500 EUR$, it can not be considered low cost [Rob14d]. At least, we need to verify whether it is possible to build a less expensive mobile base. Therefore, we have a more detailed look on these mobile bases, which mostly consist of Motors and Motor drivers.

#### Motors

There two types of motors: combustion motors and electronic motors. Unlike the *Boston Dynamics BigDog* for our setup the maintenance, repair and overhaul of combustion engines are just some of various reasons why they are less suitable. Hence, electric motors are be the better choice, but they in turn, there are different types of electric motors we are now are going to introduce.

**DC Motors** *Direct Current* (DC) motors are using a coil of wire, generating an electromagnetic field around a centered stationary set of magnets. By switching the poles of the coil (direction) or the current between on and off (speed) the shaft can be controlled. Furthermore, the total amount of current sent through the motor, the coils size and the surrounding material like electrical connections influences the mechanical characteristics of the motor[Her09]. In our case, as we are unable to provide either a lot of current (because of the usage of batteries) or use enough money to compensate that disadvantage, plain DC motors in general do not have enough torque. Furthermore, they provide relatively inaccurate movement measurement possibilities without potentiometers, because of their own momentum of inertia.
Another important point to mention is the decision between brushed and brushless DC electronic motors. Brushed DC motors use conductors between stationary and moving parts

which quickly wear themselves out, whereas brushless DC motors use rotating electromechanical magnets that need less contact to their surroundings. Furthermore, brushless usually offer increased efficiency, reduced noise and provide longer lifetime due to no erosion occuring on the brushes. During this thesis we experimented with two different DC motors shown in **Tab.7**.

| Name | maximum torque | max rpm | Voltage | Current | Geared | Price EUR |
|---|---|---|---|---|---|---|
| MakeBlock DC Motor-25 6V | 15 Ncm | 185 | 4-12V | 65mA | no | 12.36 |
| RB-35 1:30 | 60Ncm | 174 | 12V | 350mA (2A) | yes | 20.00 |

**Tab. 7:** tested DC Motors Overview

**Stepper Motors** Stepper motors are a special type of brush-less DC motors which divide a 360 degree shaft rotation in a certain number of equal steps. Current stepper motors are bipolar, which means that they consist of two separate phases containing two electronic magnets. Each run mode, like *Fullstep*, *Halfstep*, *Doublestep* and *MicroStep* offers different torques, angles and power consumptions well as they implement different timing pulses between the phases. Depending on the run mode each mentioned step in turn needs between four and eight special commands.
Below the so called holding torque the Stepper Motor would not move without any instruction. That is the reason why these motors are often used in areas of high precision like 3D printers, scanners or for mediums like like *DVD* or *Blue-Ray.*
During this thesis we evaluated two stepper motors for their usage as mobile robots engine shown in **Tab.8**.

| Name | Steps per resolution | maximum torque | max rpm | Voltage | Current | Geared | Price EUR |
|---|---|---|---|---|---|---|---|
| 28BYJ-48 | 64 | 34Ncm | 42 | 5V | 92mA | yes | 2-4 |
| NEMA 17 BiPolar | 200 | 20Ncm | 170 | 12V | 700mA | no | 13 |

**Tab. 8:** tested Stepper Motors Overview

In summary, especially the low holding torque and high power consumption while not moving prevents the usage of stepper motors for us. Furthermore, the maximum speed of this motor gets limited by the clock cycle of the used controller and therefore did not reach more than 170 rpm during our experiments. Additionally the torque dropped close to zero on top speed.

**Servo motors** Another brushless DC motor type are servo motors. As we can see in **Fig. 26** They are a combination of gears, a potentiometer and a DC motor. They usually are capable of turning 180 degrees in each direction but with modifying the gear system can be significantly enhanced to continuous rotational devices [Saw14].

Most use cases for Servo motors are exact positioning applications, like moving cameras or robot arms. Apparently, most low cost servo motors are too weak to move an UGV of our physical dimensions and properties. As we do not plan to move our depth sensor other than by moving the whole robot, they are not used in our project. Furthermore, when not adjusted properly servo motors tend to jitter continuously as they try to achieve an exact position but fail.

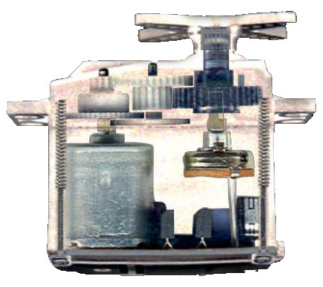


**Fig. 26:** the Servo Motor X-Ray by Darren Sawicz Princeton [Saw14]

**Gears** Using gears has a big influence on motor properties. The most common effects are a speed increase while accepting a loss in maximum torque, or vice versa, lower maximum speed decrease while gaining maximum torque. Especially in the low cost segment, these changes in motor properties can be remarkably. Furthermore, comparing geared motors with non geared motors brings misleading results and should be avoided as far as possible. For example the *RB-35* with a transmission of 1 : 30, which spins its internal motors at 6000 rpm with a tiny physical force, while finally moving geared 174 rpm with an acceptable torque of $60Ncm$.

***Pulse-width Modulation* (PWM)** DC motors can be run at different speeds by influencing the period of being turned on during the continuous switching progress. Another example for commonly used PWM is dimming LEDs which appear less bright when constantly switched on and off. The time between these two phases of being turned on and off again, is called *pulse* width and consequently PWM is a modulation technique that controls this width of pulses.
As our tests showed, unstable PWMs can be a serious problem. Exceptionally occurring on concurrent processing hardware controllers it is the the main reason why software driven GPIOs of SBCs can not be used as constant pulse clock. As their inconsistent timings, caused by computational requirements of other processes, are remarkably influenced. Furthermore, software PWMs themselves require a lot of computational power. Complementary to tests with two 64 *steps per round* stepper motors connected to the *Raspberry Pi* were surprisingly successful, but still clearly described an upper bound.
To solve the issue of software driven timings, some controllers like the *Arduino* offer special hardware PWM pins. These are connected to a dedicated electronic oscillator generating a periodic signal. An therefore can not influenced by external computational load.
One option to outsource that task further than to programmable micro controllers are custom circuits, which we examine in the next chapter.

### Motor drivers

The maximum torque of an electronic motor is contingented to the amount of available maximum power which in turn is subjected to the relation between voltage and actual total current.
Broadly speaking, with a low voltage the power flows slowly and requires a certain diameter, called current, to supply the motor sufficiently. On the other hand, a high voltage system can use a lower current for realizing enough power reaching the motor.
This simplified basic behavior of electricity explains the generally obtainable voltage current constellations especially for stepper motors. Typically, power settings for the maximum holding torque lie between $0.5A$ with more than $12V$ and $5A$ with voltages below $5V$.

Therefore, the electronic properties of motors remarkably differ from the usual conditions a SBC needs. These are mostly expecting $5V$ at $350mA$ to $900mA$. Consequently, we need different voltages in several circuits. This subsequently can come with transforming losses that can be crucially.
One way for controlling differently voltaged circuits are optocouplers. Coming with the LED technology, these optical isolators are able to transfer electronic signals like switching pulses and prevent short circuits.
To solve that task efficiently without using self constructed complex circuits a type of devices has been developed particularly over the past years: *Motor drivers.*
Beside the already mentioned, a motor driver needs to perform several other tasks we now examine.
An H-Bridge is a special electronic circuit required when moving motors, both forward and backward, by switching poles accordingly. It prevents short circuits by using four internal switches which are often graphical representable in the shape of the letter *H*. They are found in most motor driver chips in some form - mostly as integrated circuits [Wil02].
Furthermore, some motor drivers offer an abstraction of hardware PWMs. To generate different speeds other than the periodic pulse generated by the oscillator some hardware logic is required to skip certain amounts of pulses. Overall, this technique is more reliable than everything else we tested.
Another interesting point is the capability of *Inter-Integrated Circuit* (I2C). If a motor driver supports this protocol, it further can be stacked together for more complex setups. To reach each driver and motor, the bus system requires some soldered hardware connections resulting in unique addresses.
We tested two obtainable Motor driver boards with different features.

**L298N** The motor driver chip illustrated in **Fig.27** consists of two H-Bridges that are controlled by two logical signals each. Hence, it is capable of running two DC motors at the same time. During this thesis, we further used the *LN298* to run a single bipolar stepper motor.
The work load circuits can manage $2A$ of operating current, each up to a voltage of $50V$ without overheating or burning [Spa]. On the other hand, the logical power circuit processes high signals between $-0.3V$ and $7V$ consuming an for most controllers acceptable amount of current ($30mA$ max). Consequently, it allows a wide range of motor-controller combinations. Depending on the distributor this board further costs below $5EUR$ and has a good availability.

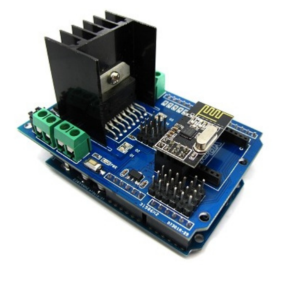


**Fig. 27:** the LN298 on a driver board [ITe11]

In contrast to the following, the *LN298* is a very simple motor driver and does not solve the issues coming with software PWM.

**Adafruit Motor Shield v2** The *Adafruit Motor Shield v2* is based on a *TB6612 MOSFET* chip that includes four separate H-Bridges. Each of its channels supports a voltage between $5V$ and $12V$ and allows $1.2A$ operating and a $3A$ maximum current during peak. Configurable by a jumper, the logic circuits can be triggered by $5V$ or $3.3V$ high signals. This makes the shield compatible to most existing *Arduino* controllers.
Furthermore, the shield integrates features like the built-in flyback diodes, which prevent damage on batteries when the UGV is physically moved without using the connected motors. Another helpful component is the fully dedicated PWM driver chip.

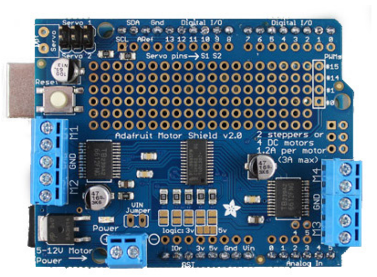


**Fig. 28:** the Adafruit Motor Shield v2 [Roy14]

The shield, shown in **Fig.28**, supports up to four bidirectional DC motors or two bipolar stepper motors. Additionally, two more $5V$ Servo Motors can be connected jitter-free. Furthermore, the motor shield can be stacked with up to 32 shields, controlling 64 stepper or 128 DC motors with a single microcontroller. This can be achieved by I2C which only requires a clock and a signal wire to be operable.
Unfortunately, the software side currently does not support I2C other than being emitted by an *Arduino*. It generally would be possible to port the code to a compatible SBC but would require more work than we were able to provide during that thesis.

## 3.3 Further extensibility

Equally to the *iRobot PackBot*, some UGVs follow the concept of modularized functionality. This often is achieved by creating autonomously working and stackable dimensional standardized units attached to a cargo bay. These units can be connected to the UGV by some kind of serial communication or without any physical connection at all using wifi, bluetooth or similar interfaces.
Consequently, creating multiple exchangeable extension sets increases the feature-set and allows fast reconfiguration during heterogeneous tasks.
A practical example for our robot would be to carry a special mesh communication device, like a portable and therefore separately powered wireless router. By that, the robot could drive to a certain position, turn it self off and keep on serving as a communication edge. When its time to return to the base station has come, the UGV turns on again by the *Ethernet wake on lan* feature. Consequently, the robot saves power and would reach higher times of service using a single battery charge.

# 4 Implementation

In this chapter we describe how to rebuild the low cost ROS compatible UGV, called ***a Mobile Sensor Robot*** (aMoSeRo), which we developed during this thesis. The suggested setup should be understood as a sample configuration of an inexpensive *base_footprint* and is the result of our various hardware tests. Combining a simple IMU, an advanced depth sensor and a basic motor control, this configuration can be later extended and improved modularly.

## 4.1 aMoSeRo Hardware

### 4.1.1 List of parts

The prices listed in **Tab.9** are indications and vary with time and supplier.

| **Amount** | **Name** | **Price per unit EUR** | **Total price EUR** |
|---|---|---|---|
| 1 | Asus Xtion Pro Live | 139.90 | 139.90 |
| 1 | CubieTruck | 88.00 | 88.00 |
| 1 | Powerbank 1x12V 2x5V USB | 45.00 | 45.00 |
| 2 | Arduino Micro | 22.67 | 45.34 |
| 2 | Motorcraft RB-35 Gear Motor 1:30 | 19.95 | 39.90 |
| 1 | Sparkfun 9DOF LSM9DS0 | 28.51 | 28.51 |
| 2 | Makeblock Track With Track Axle(40 Pack) | 8.82 | 17.64 |
| 1 | Adafruit Motor Shield v2 | 17.56 | 17.56 |
| 2 | Pololu Universial Aluminium Mounting Hub 6mm | 7.57 | 15.14 |
| 1 | Makeblock Timing Pulley 90T Blue 4-Pack | 10.00 | 10.00 |
| 2 | Plexiglass 30cm x 30cm | 5.00 | 10.00 |
| 1 | 2m aluminium angle section 90° 30mm x 30 mm | 10.00 | 10.00 |
| 1 | Metal Glue | 8.00 | 8.00 |
| 1 | Pack of various colored jumper wires | 5.00 | 5.00 |
| 2 | Half BreadBoards | 2.50 | 5.00 |
| 1 | Makeblock Threaded Shaft 4x39mm 4 Pack | 2.00 | 4.00 |
| 1 | Makeblock Bearing | 2,00 | 4,00 |
| 2 | USB-MicroUSB cables | 2.00 | 4.00 |
| 1 | Bidirectional Logic Level Converter 5V to 3.3V | 1.95 | 1.95 |
| 1 | PowerJack 1.7mm | 1.00 | 1.00 |
| | | **Total** | **499,94** |

**Tab. 9:** List of required parts

In addition to the mentioned parts, some tools to cut the aluminium and the plexiglass for example a normal and a metal saw, a $4mm$ and a $6mm$ metal drill and a cordless screwdriver are required. Furthermore, electronic parts like the IMU require to get their pins soldered, which is why a soldering bold and soldering tin should be at hand. Additionally, to sort and fasten certain parts, some wire tie and a hot glue gun are optional.

To run the robot on the software side at some point you are required to flash the operating system to the onboard memory of the *CubieTruck*. During this thesis a $4GB$ micro USB card was sufficient. Furthermore, to visualize the robots sensor data or to solve computationally intensive tasks an average *Linux* driven *Laptop* was required.
Another point to mention in order to save costs is that the official *Arduino* micros can be legally replaced by unofficial clones which usually only cost a third of the original price. The *Adafruit Motor Shield v2* further is compatible to better suitable *Arduino Uno Boards*, where it can simply be plugged onto.

### 4.1.2 Case

Physical dimensions, material and weight at an affordable price require us to construct a custom case for the robot. To rebuild our version, start by cutting the two meter aluminium angle section into four parts as shown in **Fig.29**. Arrange them for the UGV lid like shown in **Fig.31** and like shown in **Fig.32** for the corresponding bottom. After that, glue the arrangements together following the glues specific instructions. Meanwhile the glue is curing, carefully cut the glass into two pieces, like illustrated on **Fig.30**. Later hot glue them inside the bottom and lid.

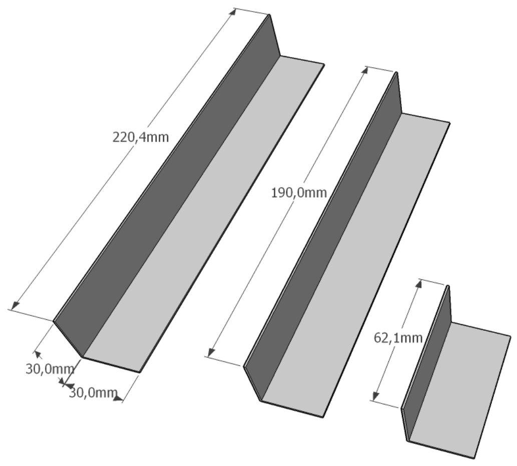


**Fig. 29**: aMoSeRo aluminium angle sections to be cut four times each

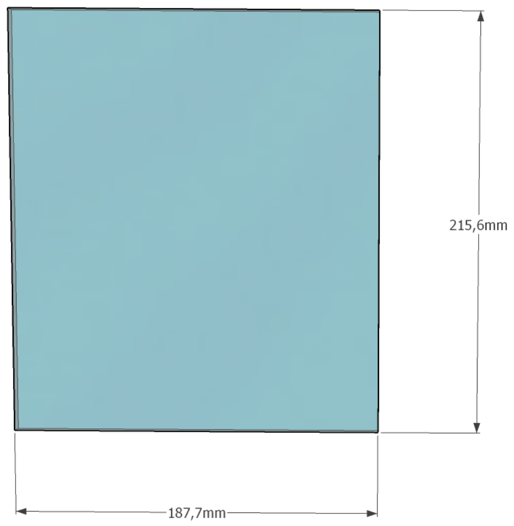


**Fig. 30**: the aMoSeRo plexiglass dimensions

To fix the motors, we drill two $6mm$ diameter holes as illustrated in **Fig.35**. Next, we insert them carefully and further fix them with hot glue or ties if required. As a next step, we attach the wheels to the motors coil using the *Pololu Universial Aluminium Mounting Hub*. After that, we drill two $4mm$ wholes into the other side, and connect the front wheels using the corresponding construction kit. Finally, we build the tracks by connection the track parts with their bolds and tightly laying them around the wheels.

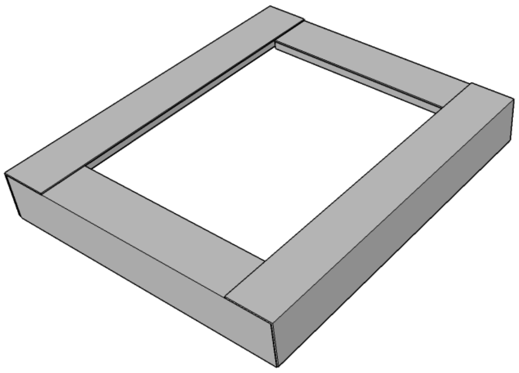


**Fig. 31:** the aMoSeRo lid without plexiglass

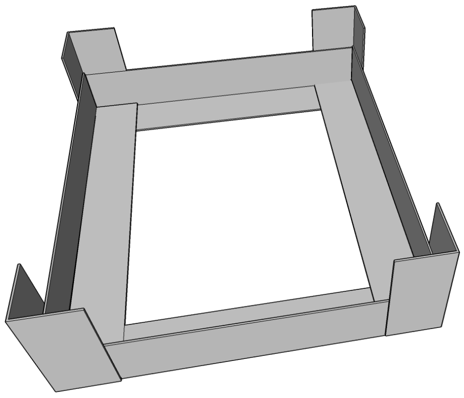


**Fig. 32:** the aMoSeRo bottom without plexiglass

### 4.1.3 Wiring schematic

#### Motor Driver

The *Adafruit Motor Shield v2* is very well documented by its producer *Adafruit*. We follow their basic tutorials and connect all parts like illustrated in **Fig.33**.

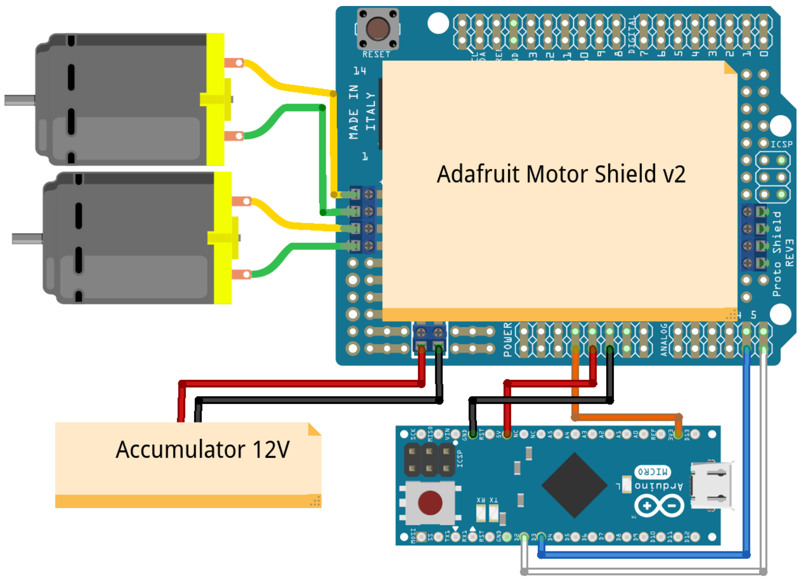


**Fig. 33:** the Adafruit Motorshield v2 Wiring created with Fritzing

#### LSM9DS0

As introduced in section 3.2.1, the *LSM9DS0* is an inexpensive 9DOF IMU. It operates at 3.3$V$ and therefore requires a special wiring circuit including a bidirectional logic level converter. This is capable of transferring the 5$V$ I2C signals to the 3.3$V$ *LSM9DS0* in both directions. It would be possible to construct a circuit of this kind on our own, but using a pre-assembled variant is easier to reproduce and also cheaper in small amounts.
When placed in the robot, the coordinate systems printed on the chip must be kept in mind. These need to fit the **REP103**. Consequently, the direction of travel corresponding to the wiring schematic **Fig.34** is right to left.

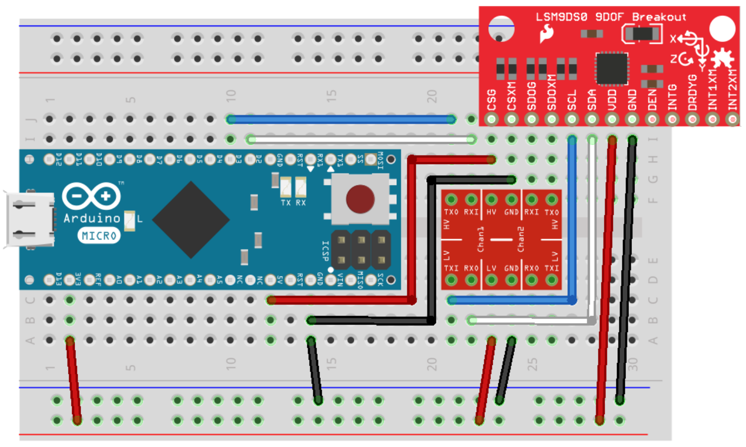


**Fig. 34**: the *Inertial Measurement Unit* (IMU) wiring created with Fritzing

### 4.1.4 Combination

We arranged all parts like the following **Fig.35** and **Fig.36** suggests.

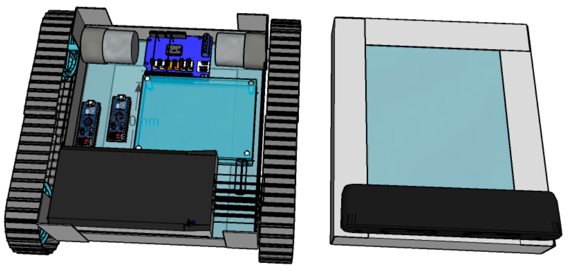


**Fig. 35**: the aMoSeRo 3D model open lid

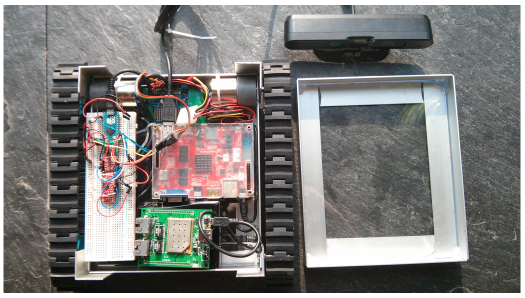


**Fig. 36**: the aMoSeRo real robot open lid

## 4.2 aMoSeRo ROS package

The aMoSeRo package is a distributed system consisting of at least two computers.
First, the robot itself which runs *Lubuntu 13.04* in combination with ROS *Hydro*. At the beginning of this thesis, this was the latest ROS version. Later, ROS got updated to ROS *Indigo* component wise, which when transferred to the robot and its ARMv7 architecture resulted in an unstable system. Additionally, the current package manager situation lead to the decision to stay on ROS *Hydro* for the aMoSeRo.
Second, the processing device is any ROS *Indigo* compatible *Ubuntu x86 Laptop* which is responsible for calculating advanced and therefore computationally intensive tasks like SLAM or for combining different odometry data. Furthermore, it is the human visualization screen and interfaces different control sensors to the robot.
The aMoSeRo package is a ROS metapackage and contains several packages which we will now delve into.

### 4.2.1 amosero_description

In order to interface the ROS *tf2* package, the aMoSeRo needs a valid URDF file. It contains frame and joint relations and therefore is the base for various tasks. As a member of *amosero_description* the file gets cross-referenced in nearly all other thesis dependent packages. These tasks include visualization, simulation, transformation, navigation and collusion avoidance.

### 4.2.2 amosero_bringup

The aMoSeRo bringup package contains all required launch files to get the robot fully operational. It therefore offers following launch files:

***xtion.launch*** starts the camera nodelet manager and publishes all available topics into the /**camera** node. Separately, the zero copy **depthimage_to_laserscan** nodelet manager gets initialized and offers a two dimensional /**scan** topic.

***motor.launch*** launches the serial communication rosnode to communicate with the *Arduino* connected to the motor shield, it subscribes the /**right_motor** and the /**left_motor** topic. Those accept values between 0 and 510 to control the motors speed and direction.

***odometry.launch*** combines all odometry related launch files and starts them together.

***tf_broadcast.launch*** starts the tf broadcasting nodes *robot_state_publisher* and *joint_state_publisher* and publishes messages in the tf tree below the *base_footprint*. Furthermore, it transforms *geometry/Twist* messages into valid motor commands by publishing the according topics as well as it calculates the /**odom** topic accordingly.

***gps.launch*** launches the nmea node and defaults to the device */dev/ttyUSB0*. Odometry related information get normalized and published into the /**vo** topic

***imu.launch*** starts the *lsm9ds0_imu_9dof* node, which manages the serial connection to the Arduino connected to the IMU and was specifically created for this thesis. The data gets published at the /**imu_data** topic.

***robot_pose_ekf.launch*** combines /**vo** and /**imu_data** with /**odom** and uses kalman filters to create the /**robot_pose_ekf/odom_combined** topic.

Most launch files use defaults but can be started with different parameters. For details, see the comments in the source files. Another point to mention is that hardware related launch files need to be started on the aMoSeRo with root rights. Furthermore, we suggest using a shell session management tool like *screen* or setting up proper system services.

### 4.2.3 amosero_teleop

The aMoSeRo teleop package's main task is generating /**cmd_vel** messages of the type *geometry/Twist* according to different input devices.

***teleop_keyboard.py*** creates the /**cmd_vel** according to keyboard inputs.

***teleop_ps3.py*** creates the /**cmd_vel** according to *Playstation 3* inputs.

***teleop_xbox.py*** creates the /**cmd_vel** according to *Microsoft Xbox* inputs.

***teleop_joystick.py*** creates the /**cmd_vel** according to the /**joy** topic.

### 4.2.4 amosero_navigation

The aMoSeRo navigation package offers following launch files.

***gmapping_demo.py*** starts *mapping_server* and initializes *move_base* before attaching to its main functionality, which are SLAM algorithms based on *openSLAM*. It uses /**odom_combined** and /**scan** as input sources and publishes *base_footprint* to /**map** *tf* relation.

***acml_launch.py*** creates local and global cost maps according to the robots dimensions and provides basic planning functionality.

The full TF Tree is shown in the appendix aMoSeRo ROS TF Tree. Furthermore, the aMoSeRo capabilities graph is illustrated in appendix aMoSeRo ROS Capabilities Graph.

### 4.2.5 amosero_viz

The aMoSeRo viz package contains default rviz configuration files, which are plain text files. Other packages are referring to this packages in different launch files. It is common practice to use a separate visualization packages for ROS robots, which easily can be explained by the behavior of collaborative tools like *git*, where user-dependend and therefore frequently changed files should be separated from the source to avoid unnecessary merges or at least confusion. This also is the main reason why we followed that principle.

## 4.3 aMoSeRo Microcontroller

#### Motor Shield

We need to transfer the *motor_shield_v2.ino* on the *Arduino* connected to the *Adafruit Motor Shield v2*. After starting the corresponding *motor.launch* file the *Arduino* subscribes to the /**left_motor** and /**right_motor** accepting normalized *Int16* values between 0 and 255 as motor speeds with direction forward and 256 to 511 with direction backwards.

#### LSM9DS0

The second *Arduino* requires to be programmed with *LSM9DS0_SimpleCSV.ino* which is located in the *Arduino* folder of the *LSM9DS0* package. The fundamental functionality equals other IMU drivers e.g. the *9DRazor*.
Unfortunately sometimes our setup randomly froze. Hence, it required a hardware reset when the data flow got stuck, which automatically gets handled by software. During this thesis it was not possible to determine if this behavior is limited on our physical devices or if other IMUs are faced with it as well.

#### Simple sensors

We further tested various simple sensors like the LM35 which is capable of recording temperature over time. The example source code, which runs on the *Arduino*, can also be found in the appendix Arduino Micro Sample temperature sensor LM35.
Moreover, we experimented with a *DHT11* Temperature sensor, a *HC-SR04 Ultrasonic Range Scanner* and a *BMP180* Barometer, which source code can be found in the *amosero_bringup* package.

# 5 Evaluation

In this chapter we are going to evaluate the robots capabilities. Therefore, the primary goal of the following experiments is to determine the limitations of aMoSeRo and research improvement possibilities.

## 5.1 aMoSeRo 2D Laser scan frame rates

An important property of an UGV is the rate of data creation. A low rate influences most higher algorithms leads to incorrect results. In our case especially the depth sensors are required to publish sufficient material to create detailed maps. Therefore the first experiment is examining the hardware limitations of the aMoSeRo.
The *Asus Xtion Pro* driver *OpenNi2* and the ROS package *openni2_camera* offers multiple run modes which can be set by *dynamic_reconfigure*. Another essential option influencing performance is the *data_skip* parameter, which allows the system to skip a certain amount of pictures the hardware produces before loading them into memory and by that remarkably reduces computational load. It can be set to an integer value between zero, which means not to skip any frames at all, and ten, leading to every eleventh frame to be processed.
The different combinations of resolutions, maximum frequencies and the *data_skip*-parameter ran on the aMoSeRo is illustrated in **Tab.10**. As it can be seen, especially the amount of frames that has to be processed per second highly influences the complete system.

| Runmode(ID) | Resolution | data_skip | CPU | avail. rate hz | operational |
|---|---|---|---|---|---|
| SXGA_30Hz(1) | 1280x1024 | - | - | not applicable (n.a.) | - |
| SXGA_15Hz(2) | 1280x1024 | - | - | n.a. | - |
| XGA_30Hz(3) | 1280x720 | - | - | n.a. | - |
| XGA_15Hz(4) | 1280x720 | - | - | n.a. | - |
| VGA_30Hz(5) | 640x480 | 10 | 95 / 80 | 1.9 | yes |
| VGA_30Hz(5) | 640x480 | 0 | 99 / 99 | 6.74 | no |
| VGA_25Hz(6) | 640x480 | 10 | 93 / 88 | 1.9 | yes |
| **VGA_25Hz(6)** | **640x480** | **2** | **89 / 88** | **8.2** | **yes** |
| VGA_25Hz(6) | 640x480 | 0 | 92 / 99 | 9.12 | no |
| QVGA_25Hz(7) | 320x240 | 10 | 65 / 60 | 2.26 | yes |
| QVGA_25Hz(7) | 320x240 | 0 | 80 / 90 | 25.00 | yes |
| QVGA_30Hz(8) | 320x240 | 10 | 70 / 52 | 2.5 | yes |
| QVGA_30Hz(8) | 320x240 | 0 | 80 / 93 | 30.0 | yes |
| QVGA_60Hz(9) | 320x240 | 10 | 87 / 70 | 4.83 | yes |
| QVGA_60Hz(9) | 320x240 | 0 | 99 / 99 | 30.24 | no |
| QQVGA_25Hz(10) | 160x120 | - | - | n.a. | - |
| QQVGA_30Hz(11) | 160x120 | - | - | n.a. | - |
| QQVGA_60Hz(12) | 160x120 | - | - | n.a. | - |

**Tab. 10:** *openni2_camera* runmodes system influence

Furthermore, higher resolutions cause higher computational demand, which is a problem when building a low cost setup. But as we will learn during the following experiment, a high

resolution of the laser scan is preferable. On the other hand a system with an average load of more than 90 percent per core can not be considered as fully operational, because when faced with this situation, the robot behaves delayed or even skips teleoperational command executions.
After various tests balancing the given parameters the best tradeoff between quality and rate was run mode with the ID six *VGA_25hz* with a *data_skip*-value between one and three. By that, data rates up to $8.2Hz$ have been reached while keeping the processors below the mentioned line of inoperability.
Furthermore, we highly recommend to use a process priority management tool like *nice* and give motor control the highest priority to achieve the same results.
This experiment showed direct correlations to the 3D PointCloud topics as the *depthimage_to_laserscan* ROS package originates its data from **/camera/depth/image**.

## 5.2 aMoSeRo 2D SLAM

The previous experiment showed how careful we need to tread the aMoSeRo setup in cases of performance. To further save resources and because the common solutions did not apply successfully by generating to much computational load, we were obligated to transfer the **/scan** topic into a separate *nodelet manager*. By that we could achieve a lazy subscription feature, that remarkably profited from *zero copy* parameter handling and only required resources when subscribed.
To start the experiment of *openSLAM*, we bring the aMoSeRo up and launch the aMoSeRo navigation package on a separate powerful computer. The second computer is required to provide all non-obligatory packages and libraries. In particular the currently non-*ARM* compatible, mostly advanced math and therefore complex to port sources need the second computer to be *x86*-compatible.
We started the robot in a square room, followed by an ordinary indoor office floor. Equipped with the 9DOF IMU, the *Asus Xtion Pro* and its calculated odometry information the robot soon revealed its demand of high data flow frequencies.
As the results illustrate in **Fig.37**, missing edges and magnetic field influences like office peripherals in combination with frequencies below $2Hz$ and a lot of spot turns lead to improvable results.
When increasing the publishing rates by the methods we mentioned in the previous experiment and furthermore slowing down turning movements my limiting their speed by software, the created maps significantly got better. This shown in **Fig.38**.

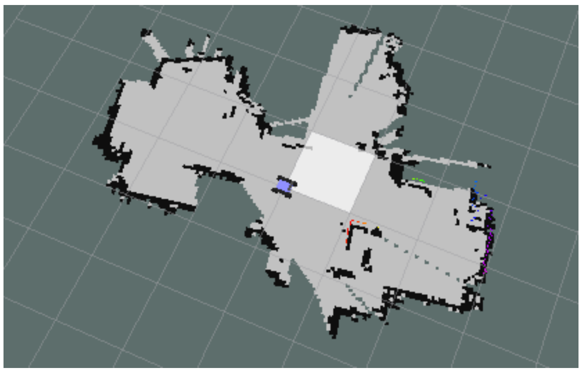


**Fig. 37:** 2D Slam low data flow frequencies

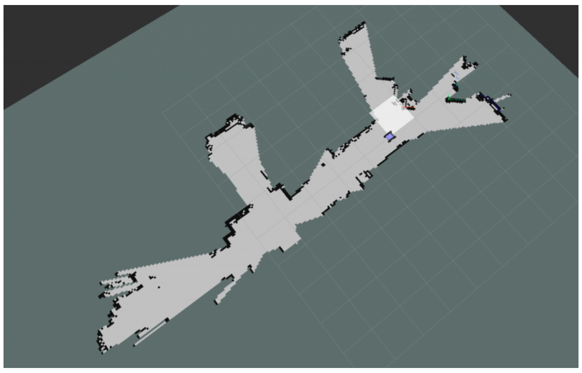


**Fig. 38:** 2D Slam high data flow frequencies

## 5.3 aMoSeRo 3D PointCloud Maps

During that thesis the requirement of creating bigger point clouds based on two dimensional slamming has been created. As shown, the aMoSeRo is currently not capable of serving high resolution 3D PointClouds at a decent rate while staying fully operational. To still fulfill the requirement, we created a special replay server. Whenever the user pulls a trigger, like in our case a keyboard input while teleoperating, a special node subscribes the **/camera/depth/image** topic for a single frame. After two to three seconds the snapshot appears in visualization and the robot can move again. Repeating the task at different locations on the map enables the low cost robot to create 3D PointClouds like they are illustrated in **Fig.39**, **Fig.40** and **Fig.41**.

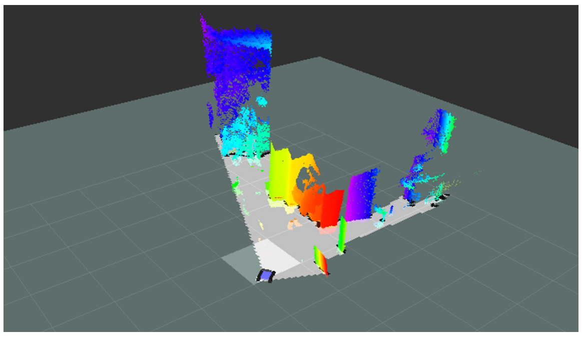


**Fig. 39**: 3D PointCloud Map based on 2D Slam with two snapshots

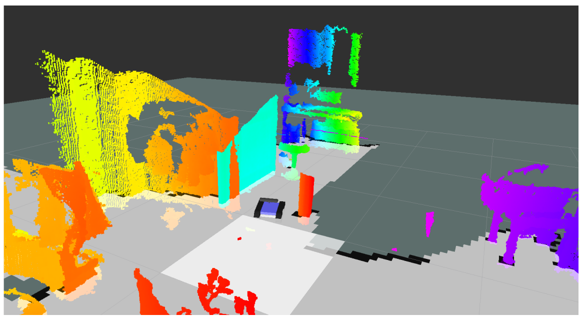


**Fig. 40**: 3D PointCloud Map based on 2D Slam with five snapshots

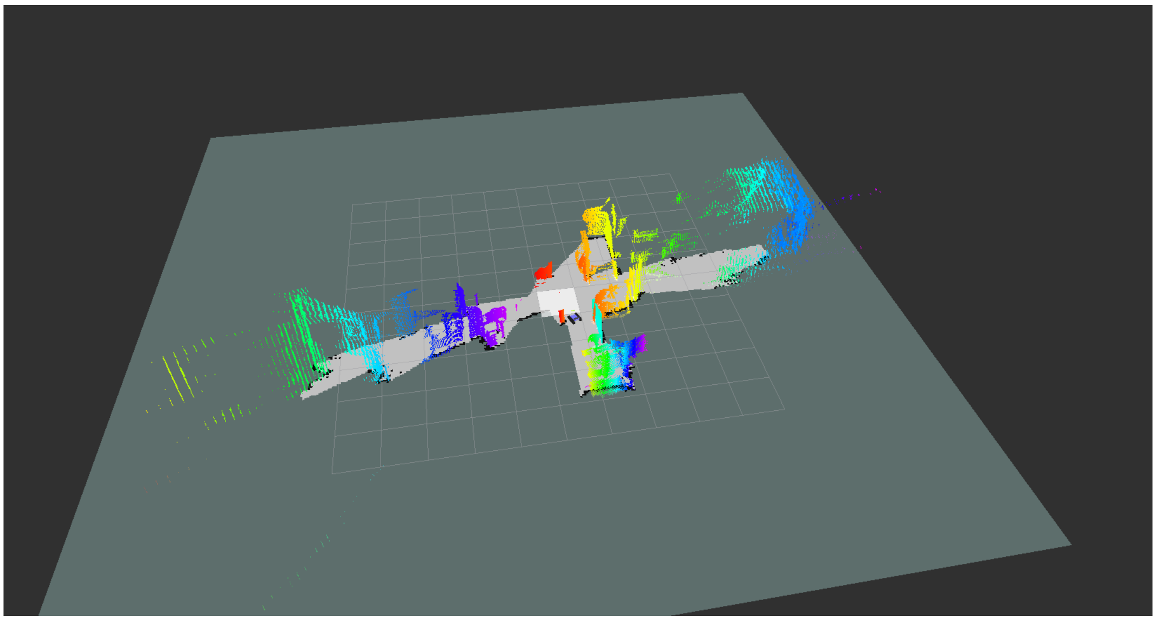


**Fig. 41**: 3D PointCloud Map based on 2D Slam with five snapshots

# 6 Conclusion

In this thesis we introduced a new way to create UGVs. In contrast to previous approaches we completely avoided to use expensive parts in all components and replaced them with low cost alternatives. We therefore were weighing the pros and cons of self-written software against using different existing *Robotic Software Environments* and found a free software solution fulfilling our needs in ROS. Moreover, we introduced existing mobile robots and their features, inspecting them for similarities and in order to find functional hardware modules. We furthermore delved into these modules, provided basic knowledge to understand their main characteristics and discussed them in the low cost context.
Subsequently, we described a step by step implementation and finally implemented a low cost UGV. The proof of concept called aMoSeRo combines smartphone technology with mass produced gaming hardware and a custom mobile base. Arising thereby, we were obligated to find creative ways to port and create drivers, ROS packages and to solve various issues, which we documented mostly separately from this thesis [Pet14].
We further evaluated the newly created robot on its capabilities and limitations and optimized certain parameters and circumstances in order to fulfil the demands of higher algorithms like SLAM and 3D PointClouds.
Another essential point to mention is that each module of the aMoSeRo can be improved with better or additional parts to fulfil the tasks it is currently not capable of. For example, when it is required to sustain a higher rate of data flow per second, you can replace the current SBC, or add another dedicated device.
In our case we additionally thought of using the aMoSeRo in a mining context. As you see in **Fig.42** water can be an issue we would need to solve with an IP67 sealed case. Furthermore rails and sludge would require stronger motors, as it is illustrated in **Fig.43**.
To summarize, our new approach successfully showed that robotics do not need to be restricted to well founded projects only. A new generation of applications, like a trail of small support robots for providing mesh network communication but with the ability to track sensor data in dangerous environments or to locate themselves, is now as well possible as the usage of a simple ROS robot that is cheaper than the educational *TurtleBot*.

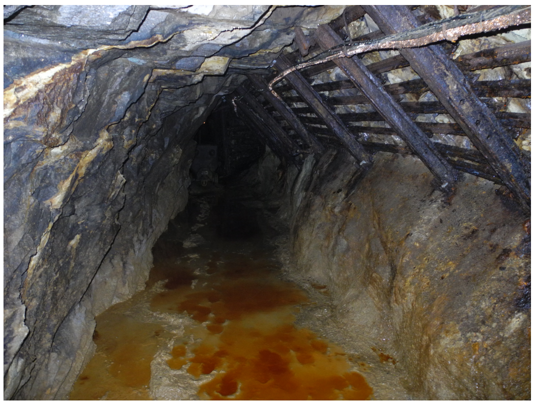


**Fig. 42:** Water in Mining - Mining RoX Project

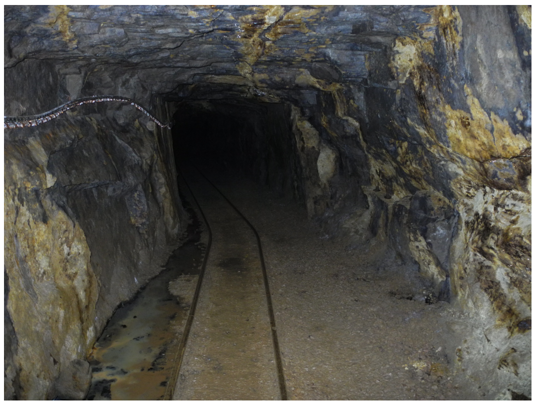


**Fig. 43:** Sludge and rails - Mining RoX Project

## Eidesstattliche Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Versicherung bezieht sich auch auf die bildlichen Darstellungen.

13. Oktober 2014 Paul Petring

## Declaration

I hereby declare that I completed this work without any improper help from a third party and without using any aids other than those cited. All ideas derived directly or indirectly from other sources are identified as such. This declaration also refers to the representation of figures and visual material.

October 13, 2014 Paul Petring

# List of Figures

1 the iRobot Roomba 880 [Rob14f] . . . 5
2 the iRobot Scooba 450 [Rob14g] . . . 5
3 the iRobot Create [Ima14] . . . 5
4 the iRobot PackBot 510 E.T [WiM14] . . . 7
5 the iRobot Negotiator [iRo14b] . . . 7
6 the Boston Dynamics *BigDog* [Qwr14] . . . 8
7 the Boston Dynamic SandFlea [Qwr14] . . . 9
8 the Boston Dynamic RHex [Qwr14] . . . 10
9 the Willow Garage TurtleBot 1 [WG14f] . . . 11
10 the Willow Garage TurtleBot 2 [WG14d] . . . 11
11 the Clearpath Robotics Husky [Ack12] . . . 12
12 the Willow Garage PR2 [Raz11] . . . 12
13 the Carnegie Mellon Groundhog [CMU05] . . . 13
14 Carnegie Mellon CaveCrawler . . . 13
15 the TU Darmstadt Team Hector Rescue Robots [Dar12] . . . 14

16 ROS a simple robot TF tree . . . 22
17 ROS a simple robot 3D model . . . 22

18 Functional modules of UGVs . . . 25
19 the HC-SR04 [Aim14] . . . 25
20 the Hokuyo's URG-04LX-UG01 [Ha14] . . . 26
21 Microsoft Kinect . . . 27
22 Asus Xtion Pro Live . . . 27
23 LeapMotion Device . . . 28
24 LeapMotion Visualization . . . 28
25 the Arduino Micro [Ard14] . . . 36
26 the Servo Motor X-Ray by Darren Sawicz Princeton [Saw14] . . . 38
27 the LN298 on a driver board [ITe11] . . . 39
28 the Adafruit Motor Shield v2 [Roy14] . . . 40

29 aMoSeRo aluminium angle sections to be cut four times each . . . 42
30 the aMoSeRo plexiglass dimensions . . . 42
31 the aMoSeRo lid without plexiglass . . . 43
32 the aMoSeRo bottom without plexiglass . . . 43
33 the Adafruit Motorshield v2 Wiring created with Fritzing . . . 43
34 the *Inertial Measurement Unit* (IMU) wiring created with Fritzing . . . 44
35 the aMoSeRo 3D model open lid . . . 44
36 the aMoSeRo real robot open lid . . . 44

37 2D Slam low data flow frequencies . . . 48
38 2D Slam high data flow frequencies . . . 48
39 3D PointCloud Map based on 2D Slam with two snapshots . . . 49
40 3D PointCloud Map based on 2D Slam with five snapshots . . . 49

41 3D PointCloud Map based on 2D Slam with five snapshots . . . . . . . . . . . 49

42 Water in Mining - Mining RoX Project . . . . . . . . . . . . . . . . . . . . . . . 50
43 Sludge and rails - Mining RoX Project . . . . . . . . . . . . . . . . . . . . . . . 50

# List of Tables

1 the Cyberbotics - Webbots versions and features overview [Cyb14b] . . . . . . 19

2 Hokuyo's Laser-Scanner price range bounds . . . . . . . . . . . . . . . . . . . . 26
3 *Inertial Measurement Unit* (IMU) Overview . . . . . . . . . . . . . . . . . . . . 30
4 Teleoperation controller Overview . . . . . . . . . . . . . . . . . . . . . . . . . 31
5 Battery technologies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
6 Single Board Computer Overview . . . . . . . . . . . . . . . . . . . . . . . . . . 34
7 tested DC Motors Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
8 tested Stepper Motors Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

9 List of required parts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

10 *openni2_camera* runmodes system influence . . . . . . . . . . . . . . . . . . . . 47

# Bibliography

[Ack12] Evan Ackerman. Huskies on mars? university of toronto developing planetary rover - ieee spectrum. http://spectrum.ieee.org/automaton/robotics/diy/huskies-on-mars-university-of-toronto-developing-planetary-rover, 09 2012. (Visited on 08/10/2014).

[Ack13] Evan Ackerman. Interview: Turtlebot inventors tell us everything about the robot - ieee spectrum. http://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot, 03 2013. (Visited on 07/28/2014).

[Ada10] Adafruit.com. We have a winner – open kinect driver(s) released – winner will use usd 3k for more hacking – plus an additional usd 2k goes to the eff! https://www.adafruit.com/blog/2010/11/10/we-have-a-winner-open-kinect-drivers-released-winner-will-use-3k-for-more-hacking-plus-an-additional-2k-goes-to-the-eff/, November 2010.

[Aim14] Aimagin. Hc-sr04 ultrasonic sensor. https://www.aimagin.com/hc-sr04-ultrasonic-sensor.html, 2014. (Visited on 09/17/2014).

[Ard14] Arduinocc. Arduino - arduinoboardmicro. http://arduino.cc/en/Main/arduinoBoardMicro, 09 2014. (Visited on 09/16/2014).

[Cle] ClearpathRobotics. Clearpath robotics - rugged all-terrain mobile robots for research. http://www.clearpathrobotics.com/. (Visited on 07/28/2014).

[CMU14] Carnegie Mellon University CMU. Cave crawler. https://www.cs.cmu.edu/~groundhog/robots_cc.html#overview, 14. (Visited on 07/27/2014).

[CMU02] Sebastian Thrun Carnegie Mellon University CMU. 3d mapping. https://www.cs.cmu.edu/~thrun/3D/mines/groundhog/, 2002. (Visited on 07/31/2014).

[CMU05] Carnegie Mellon University CMU. Groundhog. https://www.cs.cmu.edu/~groundhog/robots_ghog.html, 2005. (Visited on 07/27/2014).

[Cov14] Coverity. Coverity scan report finds open source software quality outpaces proprietary code for the first time - coverity. http://www.coverity.com/press-releases/coverity-scan-report-finds-open-source-software-quality-outpaces-proprietary-code-for-the-first-time/, 2014. (Visited on 10/12/2014).

[CTHJMB05] B. A. Collet T. H. J. MacDonald and Gerkey B. Player 2.0: Toward a practical robot programming framework. *Proceedings of the Australasian Conference on Robotics and Automation*, 2005.

[CYB14a] 2014 Cyberbotics Ltd. CYB. Webots: robot simulator - overview. http://www.cyberbotics.com/overview, 2014. (Visited on 07/27/2014).

[Cyb14b] Cyberbotics. Webots: robot simulator - documentation - user guide. `http://www.cyberbotics.com/guide.pdf`, 2014. (Visited on 08/16/2014).

[CyP14] CyPhyWorks. Cyphy works | leadership team. `http://cyphyworks.com/about/leadership/?bio=helen-greiner`, 2014. (Visited on 08/05/2014).

[Dar12] Hector TU Darmstadt. Team hector darmstadt | hector darmstadt - heterogeneous cooperating teams of robots. `http://www.gkmm.tu-darmstadt.de/rescue/`, 2012. (Visited on 07/31/2014).

[Dar14] CARM TU Darmstadt. Overview | research training group. `http://www.gkmm.tu-darmstadt.de/`, 2014. (Visited on 07/31/2014).

[D.E01] Saranli U. Buehler M. Koditschek D.E. Rhex: A simple and highly mobile hexapod robot. `www.rhex.web.tr/saranli_buehler_koditschek.ijrr2001.pdf`, 2001. (Visited on 08/10/2014).

[Dey08] Travis Deyle. Sick laser rangefinder (lidar) disassembled | hizook. `http://www.hizook.com/blog/2008/12/15/sick-laser-rangefinder-lidar-disassembled`, 12 2008. (Visited on 07/27/2014).

[DFG14] DFG. Dfg - deutsche forschungsgemeinschaft. `http://www.dfg.de/`, 2014. (Visited on 07/31/2014).

[Dic] Oxford Dictionary. robot: definition of robot in oxford dictionary (british and world english). `http://www.oxforddictionaries.com/definition/english/robot?q=robot`. (Visited on 07/28/2014).

[Dic14] LLC Dictionary.com. Laser | define laser at dictionary.com. `http://dictionary.reference.com/browse/laser`, Jul 2014. (Visited on 07/27/2014).

[Dyn12] Boston Dynamics. Rhex rough-terrain robot - youtube. `https://www.youtube.com/watch?v=ISznqY3kESI`, 03 2012. (Visited on 08/10/2014).

[Dyn13] Boston Dynamics. Rhex rough - terrain robot datasheet. `http://www.bostondynamics.com/img/RHex%20Datasheet%20v1_0.pdf`, 2013. (Visited on 08/10/2014).

[Eng13] Engadget.com. Asus partners up with leap motion, pcs with 3d motion control to debut in 2013. `http://www.engadget.com/2013/01/03/asus-leap-motion-partnership/`, 4 2013. (Visited on 07/27/2014).

[Eng14] Engadget.com. irobot create: Roomba hacking for the everyman. `http://www.engadget.com/2006/11/29/irobot-create-roomba-hacking-for-the-everyman/`, 2014. (Visited on 07/27/2014).

[fCMUC14] Anne Watzman for Carnegie Mellon University CMU. Autonomous "groundhog" maps mathies coal mine. `https://www.cmu.edu/cmnews/030625/030625_minemap.html`, 2014. (Visited on 07/31/2014).

[Giz08] Emily Clark GizMag. irobot negotiator civil response robot. `http://www.gizmag.com/irobot-negotiator/9781/`, 08 2008. (Visited on 08/09/2014).

[GLHF11] N. Michel O. Guyot L. Heiniger and Rohrer F. Teaching robotics with an open curriculum based on the e-puck robot, simulations and competitions. *Proceedings of the 2nd International Conference on Robotics in Education September 15-16 Vienna*, 2011. Retrieved February 21, 2012, from (http://www.cyberbotics.com/publications/RiE2011.pdf).

[GRK14] GRK1362. Technische universitaet darmstadt - graduiertenkolleg grk1362 · github. `https://github.com/tu-darmstadt-ros-pkg`, 2014. (Visited on 07/31/2014).

[Ha14] Hokuyo-aut.jp. Hokuyo urg-04lx-ug01. `https://www.hokuyo-aut.jp/02sensor/07scanner/img/urg_04lx_ug01_top.jpg`, 09 2014. (Visited on 09/08/2014).

[Her09] S. Herman. *Industrial Motor Control*. Delmar Cengage Learning, 2009.

[iCi14a] iRobot Corporation iRo. irobot 510 packbot battle-tested, mission-configurable robot. `http://www.irobot.com/us/learn/defense/packbot.aspx`, 2014. (Visited on 07/27/2014).

[iCi14b] iRobot Corporation iRo. irobot roomba vacuum cleaning robot. `http://www.irobot.com/us/learn/home/roomba.aspx`, 2014. (Visited on 07/27/2014).

[iCi14c] iRobot Corporation iRo. irobot scooba floor scrubbing robot. `http://www.irobot.com/us/learn/home/scooba.aspx`, 2014. (Visited on 07/27/2014).

[IEE09] IEEE. Sand flea jumping robot headed to afghanistan - ieee spectrum. `http://spectrum.ieee.org/automaton/robotics/military-robots/sand-flea-jumping-robot-headed-to-afghanistan`, 11 2009. (Visited on 08/10/2014).

[IEE12] IEEE. Boston dynamics sand flea robot demonstrates astonishing jumping skills - ieee spectrum. `http://spectrum.ieee.org/automaton/robotics/military-robots/boston-dynamics-sand-flea-demonstrates-astonishing-jumping-skills`, 03 2012. (Visited on 08/10/2014).

[IFR12] IFR. Service robots - ifr international federation of robotics. `http://www.ifr.org/service-robots/`, 2012. (Visited on 08/08/2014).

[Ima14] Imageg.net. pirbt-3426567v380.png (380×347). `http://irbt.imageg.net/graphics/product_images/pIRBT-3426567v380.png`, 2014. (Visited on 08/13/2014).

[Ini14] Open Source Initiative. The open software license 3.0 (osl-3.0) | open source initiative. `http://opensource.org/licenses/OSL-3.0`, 2014. (Visited on 09/15/2014).

[iPi13] iPiSoft. Depth sensors comparison - ipisoft wiki. `http://wiki.ipisoft.com/Depth_Sensors_Comparison`, 03 2013. (Visited on 08/05/2014).

[IPI14] IPI. industrial perception, inc. http://www.industrial-perception.com/, 2014. (Visited on 08/10/2014).

[iRo90] iRobot. Connecting the world - irobot le and irobot co-worker. http://www.irobot.com/filelibrary/ppt/corp/cool_stuff_ppt/img13.html, 1990. (Visited on 08/08/2014).

[iRo14a] iRobot. irobot: Our history. http://www.irobot.com/us/Company/About/Our_History.aspx, 2014. (Visited on 08/04/2014).

[iRo14b] iRobot. Saving lives - irobot negotiator. http://www.irobot.com/filelibrary/ppt/corp/cool_stuff_ppt/img22.html, 2014. (Visited on 08/08/2014).

[iSu07] iSuppli. irobot create contest rules -. http://www.tomshardware.com/reviews/irobot-contest-rules-20050515,1601.html, 03 2007. (Visited on 08/08/2014).

[ITe11] ITead. L298n h-bridge motor driver shield : Motomama | itead intelligent systems blog. http://blog.iteadstudio.com/ln298-h-bridge-moto-driver-shield-motomama/, 03 2011. (Visited on 09/17/2014).

[Kal60] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960. volume 82, pages 35–45, doi = 10.1115/1.3662552.

[Kue11] Thomas Kuehn. The kinect sensor platform. *Advances in Media Technology*, 2011. Retrieved February 21, 2012, from (http://www.lmt.ei.tum.de/courses/hsmt/proceedings/pdf/ss2011/01Kinect.pdf).

[Lin14] Admin Linkedcxoforum.com. A qna with irobot's colin angle, the ceo of the only consumer robotics company that matters | linked cxo forum. http://www.linkedcxoforum.com/a-qa-with-irobots-colin-angle-the-ceo-of-the-only-consumer-robotics-company-that-matters/, 06 2014. (Visited on 08/08/2014).

[MF13] Aaron Martinez and Enrique Fernández. *Learning ROS for Robotics Programming*. Packt Publishing, 9 2013.

[Mic04] O. Michel. Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 2004. Retrieved February 21, 2012, from (http://www.cyberbotics.com/publications/ars.pdf) Vol. 1, Num. 1, pages 39-42.

[Mic12a] Microsoft. Ccr introduction. http://msdn.microsoft.com/en-us/library/bb648752.aspx, 2012. (Visited on 08/15/2014).

[Mic12b] Microsoft. Dss introduction. http://msdn.microsoft.com/en-us/library/bb483056.aspx, 2012. (Visited on 08/15/2014).

[Min14] LEGO Mindstorms. Lego.com mindstorms. http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com, 2014. (Visited on 08/05/2014).

[Nav14] R Nave. Description of motion. http://hyperphysics.phy-astr.gsu.edu/hbase/mot.html, January 2014.

[Ng07] Morgan Quigley Eric Berger Andrew Y. Ng. Stair: Hardware and software architecture, aaai 2007 robotics workshop. *Proceedings of the Australasian Conference on Robotics and Automation*, 2007.

[Ng10] Morgan Quigley Brian Gerkey Ken Conley Josh Faust Tully Foote Jeremy Leibs Eric Berger-Rob Wheeler Andrew Ng. Ros: an open-source robot operating system. http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf, 04 2010.

[NYT13] NYTimes. Google adds to its menagerie of robots - nytimes.com. http://www.nytimes.com/2013/12/14/technology/google-adds-to-its-menagerie-of-robots.html?_r=1&, 12 2013. (Visited on 08/04/2014).

[O'K13] Jason M. O'Kane. *A Gentle Introduction to ROS*. Independently published, October 2013. Available at http://www.cse.sc.edu/~jokane/agitr/.

[Ols08] Stefanie Olsen. Life after google, with millions - cnet news. http://news.cnet.com/Life-after-Google,-with-millions---page-2/2100-1030_3-6226900-2.html, 01 2008. (Visited on 07/29/2014).

[On14] National Oceanic and Atmospheric Administration noaa.gov. What is lidar? http://oceanservice.noaa.gov/facts/lidar.html, 06 2014. (Visited on 07/27/2014).

[one14] one.laptop.org. about the project mission. http://one.laptop.org/about/mission, 2014.

[OSR13] OSRF. Osrf - ros @ osrf. http://osrfoundation.org/blog/ros-at-osrf.html, 02 2013. (Visited on 08/17/2014).

[OSR14] OSRF. Osrf - people. http://www.osrfoundation.org/people/#board, 2014. (Visited on 08/05/2014).

[Pet14] Paul Petring. defendtheplanet.net. http://defendtheplanet.net/, 05 2014. (Visited on 10/11/2014).

[Pla12] Player. Playerhistory - the player project. http://playerstage.sourceforge.net/wiki/PlayerHistory, 03 2012. (Visited on 08/05/2014).

[PR214a] Willow Garage PR2. Join the community | willow garage. https://www.willowgarage.com/pages/pr2/pr2-community, 08 2014. (Visited on 08/04/2014).

[PR214b] Willow Garage PR2. Order a pr2 | willow garage. https://www.willowgarage.com/pages/pr2/order, 08 2014. (Visited on 08/04/2014).

[Pre10] Oxford University Press, editor. *New Oxford American Dictionary*. Oxford University Press, 3 edition, 9 2010.

[Qwr14] Qwrt.ru. qwrt.ru google / google, darpa, boston dynamics. http://www.qwrt.ru/news/1139, 2014. (Visited on 08/08/2014).

[Rai10] Marc Raibert. Bigdog, the rough-terrain robot - youtube. https://www.youtube.com/watch?v=-Bi-tPO0OPs, 05 2010. (Visited on 08/10/2014).

[Raz11] RazorRobotics. Pr2 - white background. https://secure.flickr.com/photos/razorrobotics/6187485002/, 09 2011. (Visited on 09/15/2014).

[Res14] ROS Robocup Rescue. robocup_rescue - ros wiki. http://wiki.ros.org/robocup_rescue, 2014. (Visited on 07/31/2014).

[Rob14a] Robocup2014. Schedule/results. http://www.robocup2014.org/?page_id=36, 2014. (Visited on 08/10/2014).

[Rob14b] Unbounded Robotics. Unbounded robotics | about us. http://unboundedrobotics.com/about-us/, 2014. (Visited on 08/05/2014).

[Rob14c] Unbounded Robotics. Unbounded robotics | ubr-1. http://unboundedrobotics.com/ubr-1/, 2014. (Visited on 08/05/2014).

[Rob14d] Robotnik. Kobuki base. http://www.robotnikstore.com/robotnik/2219336/kobuki-base.html, 09 2014. (Visited on 09/11/2014).

[Rob14e] Robotnik. Robotnik. http://robotnik.es/en/, 07 2014. (Visited on 07/29/2014).

[Rob14f] RobotVacuumHut. irobot roomba 880 vacuum cleaning robot especially for pets and allergies -. http://robotvacuumhut.com/irobot-roomba-880-vacuum-cleaning-robot-especially-pets-allergies/, 2014. (Visited on 08/08/2014).

[Rob14g] RobotVacuumHut. irobot scooba 450 floor scrubbing robot -. http://robotvacuumhut.com/irobot-scooba-450-floor-scrubbing-robot/, 2014. (Visited on 08/08/2014).

[ROS14] ROS. Ros/introduction - ros wiki. http://wiki.ros.org/ROS/Introduction, 08 14. (Visited on 08/16/2014).

[Ros99] Winn Rosch. *Hardware Bible Fifth Edition*. Que, June 1999.

[Roy14] Dan Royer. New product: Adafruit motor shield v2marginally clever. https://www.marginallyclever.com/blog/2013/08/new-product-adafruit-motor-shield-v2/, 2014. (Visited on 09/17/2014).

[RPi14a] Raspberry Pi Inc. RPi. Raspberry pi. http://www.raspberrypi.org/, 2014. (Visited on 07/27/2014).

[RPi14b] Raspberry Pi Inc. RPi. Two million! | raspberry pi. http://www.raspberrypi.org/two-million/, 11 2014. (Visited on 07/27/2014).

[Saw14] Darren Sawicz. https://www.princeton.edu/ mae412/text/ntrak2002/292-302.pdf. https://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf, 2014. (Visited on 09/16/2014).

[Sha04] William Shakespeare. *As You Like It (Folger Shakespeare Library)*. Simon and Schuster, 7 2004.

[SP76] Cynthia J. Solomon and Seymour Papert. A case study of a young child doing turtle graphics in LOGO. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, AFIPS '76, pages 1049–1056, New York, NY, USA, 1976. ACM.

[Spa] Sparkfun. Datasheet l298_h_bridge.pdf. https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf. (Visited on 09/17/2014).

[Tec13] Techcrunch.com. Leap motion controller ship date delayed until july 22, due to a need for a larger, longer beta test | techcrunch. http://techcrunch.com/2013/04/25/leap-motion-controller-ship-date-delayed-until-july-22-due-to-a-need-for-a-larger-longer-beta-test/, 7 2013. (Visited on 07/27/2014).

[Tec14] Suitable Technologies. Suitable technologies beam remote presence system. https://www.suitabletech.com/, 2014. (Visited on 08/10/2014).

[tJ08] Janice Karin thefutureofthings JK. irobot negotiator the future of things science and technology of tomorrow. http://thefutureofthings.com/6195-irobot-negotiator/, 10 2008. (Visited on 08/09/2014).

[Uni13] Malmo University. Arduino faq – with david cuartielles | medea. http://medea.mah.se/2013/04/arduino-faq/, 04 2013. (Visited on 09/16/2014).

[WG10] Willow Garage WG. Comic: Reinventing the wheel | willow garage. http://www.willowgarage.com/blog/2010/04/27/reinventing-wheel, 04 2010. (Visited on 09/18/2014).

[WG14a] Willow Garage WG. Build - turtlebot. http://www.turtlebot.com/build.html, 2014. (Visited on 07/27/2014).

[WG14b] Willow Garage WG. Hardware specs | willow garage. https://www.willowgarage.com/pages/pr2/specs, 2014. (Visited on 07/27/2014).

[WG14c] Willow Garage WG. Home - turtlebot. http://www.turtlebot.com/, 2014. (Visited on 07/27/2014).

[WG14d] Willow Garage WG. Overview | willow garage. https://www.willowgarage.com/turtlebot, 2014. (Visited on 07/29/2014).

[WG14e] Willow Garage WG. Overview | willow garage. https://www.willowgarage.com/pages/pr2/overview, 2014. (Visited on 07/27/2014).

[WG14f] Willow Garage WG. Turtlebot-front-640w.png (640×418). https://www.willowgarage.com/sites/default/files/robots_turtlebot/TurtleBot-Front-640w.png, 07 2014. (Visited on 07/28/2014).

[WG14g] Willow Garage WG. Willow garage is changing | willow garage. http://www.willowgarage.com/blog/2013/02/11/willow-garage-changing, 02 2014. (Visited on 08/10/2014).

[WG14h] Willow Garage WG. Willow garage selects clearpath robotics to service and support the pr2 robot through 2016 | willow garage. http://www.willowgarage.com/blog/2014/01/17/willow-garage-selects-clearpath-robotics-service-and-support-pr2-robot-through-2016, 01 2014. (Visited on 08/10/2014).

[Wil02] Al Williams. *Microcontroller Projects Using the Basic Stamp 2nd Edition.* CRC Press, 2 edition, 2 2002.

[WiM14] Wiki Media WiMe. File:irobot packbot 510 e.t..jpg - wikimedia commons. https://commons.wikimedia.org/wiki/File:IRobot_PackBot_510_E.T..JPG, 2014. (Visited on 07/27/2014).

[Wir10a] Wired. Control a 3-d–mapping robot with gestures? just add kinect | gadget lab | wired. http://www.wired.com/2010/11/gesture-controlled-3d-mapping-robot-just-add-kinect, 2010. (Visited on 08/08/2014).

[Wir10b] Wired.com. The wired interview: irobot ceo colin angle | business | wired. http://www.wired.com/2010/10/colin-angle-irobot-ceo/all/1, 10 2010. (Visited on 07/27/2014).

[WSJ14] WSJ. 50,000 usd robot seeks room, board—and programming - personal tech news - wsj. http://blogs.wsj.com/personal-technology/2014/04/18/50000-robot-seeks-room-board-and-programming/, 04 2014. (Visited on 09/08/2014).

[Zag14] ZagrosRobotics. irobot create programmable robot. http://www.zagrosrobotics.com/shop/item.aspx?itemid=712, 2014. (Visited on 08/21/2014).

## Acronyms

**3D** three dimensional

**AGV** *Automated Guided Vehicle*

**aMoSeRo** ***a Mo****bile* ***S****ensor* ***Ro****bot*

**API** *application programming interface*

**AUVSI** *Association for Unmanned Vehicle Systems International*

**BSD** *Berkeley Software Distribution*

**CCR** *Concurrency and Coordination Runtime*

**CI** *Continuous Integration*

**CPU** *Central Processing Unit*

**CSAIL** *Computer Science and Artificial Intelligence Laboratory*

**DARPA** *Defense Advanced Research Projects Agency*

**DC** *Direct Current*

**DIY** *Do it yourself*

**DFG** *Deutsche Forschungsgemeinschaft* [DFG14]

**DOF** *Degrees of Freedom*

**DSS** *Decentralized Software Service*

**GPIO** *General-Purpose Input/Output*

**GPL** *GNU General Public License*

**GPS** *Global Positioning System*

**GUI** *Graphical User Interface*

**Hector** *Heterogeneous Cooperating Team of Robots*

**IFR** *International Federation of Robotics*

**IDE** *Integrated Development Environment*

**IMU** *Inertial Measurement Unit*

**I2C** *Inter-Integrated Circuit*

**LAMI** *Laboratoire de Micro-Informatique*

**Lidar** *Light radar*

**Laser** *light amplification by stimulated emission of radiation* [Dic14]

**LS3** *Legged Squad Support System*

**MRDS** *Microsoft Robotics Developer Studio*

**MIT** *Massachusetts Institute of Technology*

**mph** miles per hour

**n.a.** not applicable

**NOAA** *National Oceanic and Atmospheric Administration*

**OCU** *Operator Control Unit*

**OLPC** *One Laptop Per Child Program* [one14]

**OpenCV** *Open Source Computer Vision*

**OS** *Operating System*

**OSL** *Open Software License* [Ini14]

**OSRF** *Open Source Robotics Foundation*

**PWM** *Pulse-width Modulation*

**RADAR** *RAdio Detection And Ranging*

**RAM** *Random Access Memory*

**REF** *Rapid Equipment Force*

**REP** *ROS Enhancement Proposal*

**REST** *Representational State Transfer*

**ROI** *Roomba Open Interface*

**ROS** *Robot Operating System*

**rpm** *rounds per minute*

**SBC** *Single Board Computer*

**SAIL** *Stanford Artificial Intelligence Laboratory*

**SLAM** *Simultaneous Localization and Mapping*

**SPARK** *Starter Programs for the Advancement of Robotics Knowledge*

**STAIR** *STanford Artificial Intelligence Robot*

**STEM** *Science, Technology, Engineering and Math*

**SoC** *System on a Chip*

**SPI** *Serial Peripheral Interface*

**TF** *Transformation*

**TDD** *Test Driven Development*

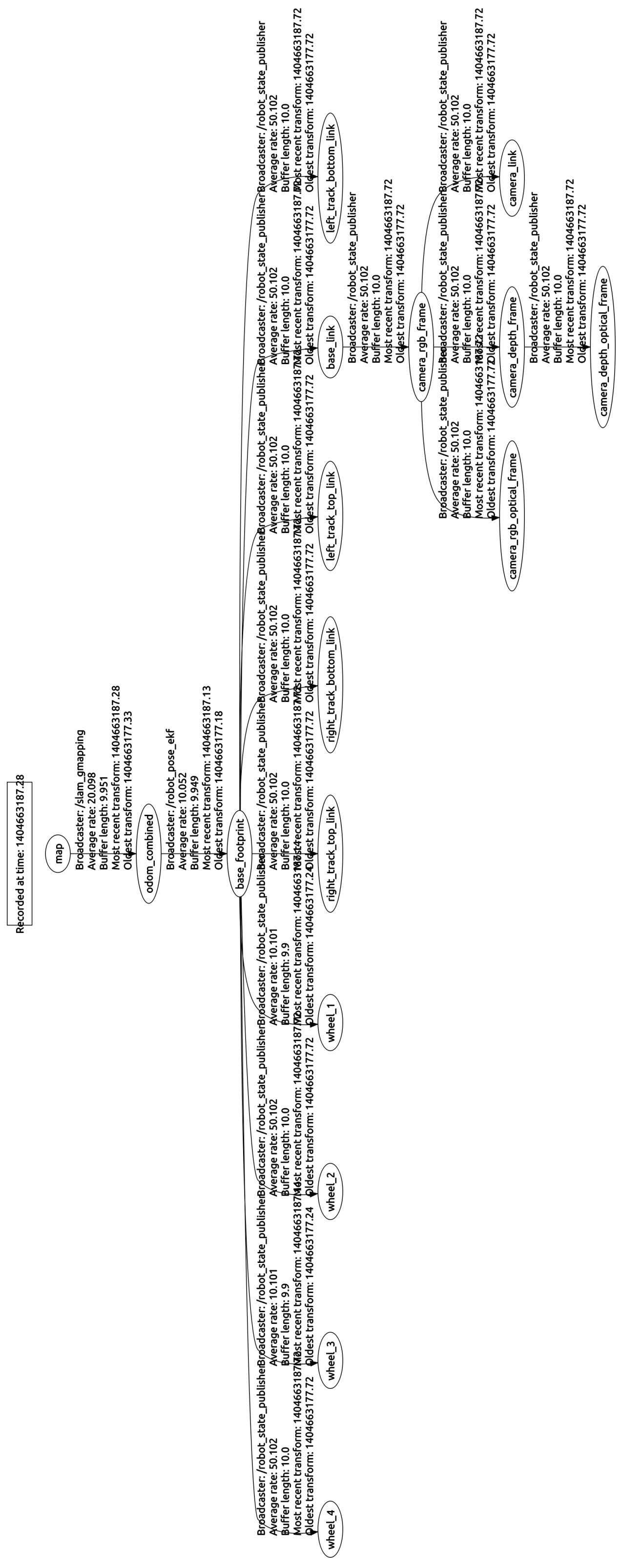
|  |  |
|---|---|
| **UART** | *Universal Asynchronous Receiver/Transmitter* |
| **UGV** | *Unmanned Ground Vehicle* |
| **URDF** | *Unified Robot Description Format* |
| **USB** | *Universal Serial Bus* |
| **VOR** | *Visual Object Recognition* |
| **VPL** | *Visual Programming Language* |
| **vSLAM** | *visual Simultaneous Localization and Mapping* |
| **XACRO** | *XML Macros* |
| **XML** | *eXtensible Markup Language* |

# Appendix

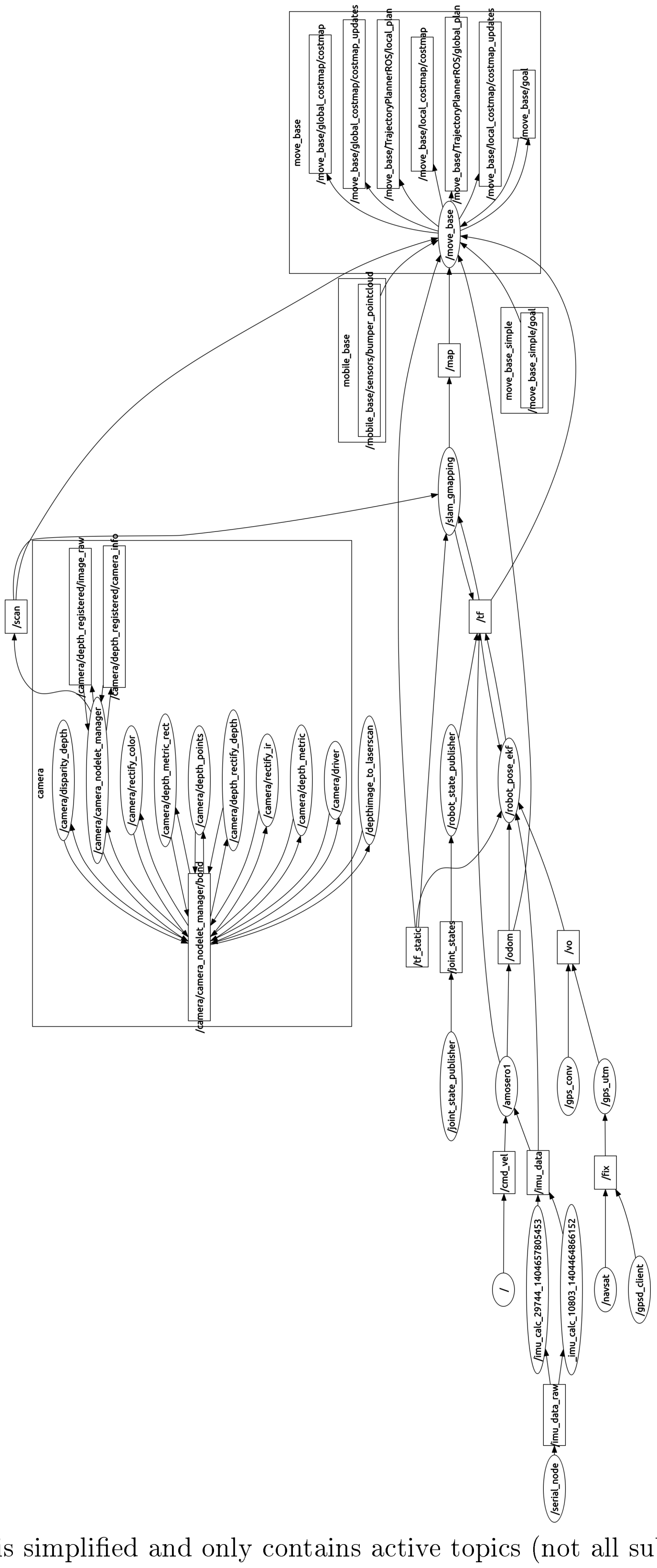
## Arduino Micro Sample temperature sensor LM35

```
float temp;
int tempPin = 4;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  temp = analogRead(tempPin);
  temp = temp * 0.48828125;
  Serial.print("TEMPRATURE = ");
  Serial.print(temp);
  Serial.print("*C");
  Serial.println();
  delay(1000);
}
```

## aMoSeRo ROS TF Tree



## aMoSeRo ROS Capabilities Graph


move_base
/move_base/global_costmap/costmap
/move_base/global_costmap/costmap_updates
/move_base/TrajectoryPlannerROS/local_plan
/move_base/local_costmap/costmap
/move_base/TrajectoryPlannerROS/global_plan
/move_base/local_costmap/costmap_updates
/move_base/goal
/move_base
mobile_base
/mobile_base/sensors/bumper_pointcloud
/map
move_base_simple
/move_base_simple/goal
/slam_gmapping
/scan
/camera/depth_registered/image_raw
/camera/depth_registered/camera_info
/tf
camera
/camera/disparity_depth
/camera/camera_nodelet_manager
/camera/rectify_color
/camera/depth_metric_rect
/camera/depth_points
/camera/depth_rectify_depth
/camera/rectify_ir
/camera/depth_metric
/camera/driver
/depthimage_to_laserscan
/robot_state_publisher
/robot_pose_ekf
/camera/camera_nodelet_manager/bond
/tf_static
/joint_states
/odom
/vo
/joint_state_publisher
/amosero1
/gps_conv
/gps_utm
/cmd_vel
/imu_data
/fix
/
/imu_calc_29744_1404657805453
/imu_calc_10803_1404464866152
/navsat
/gpsd_client
/imu_data_raw
/serial_node

The graph is simplified and only contains active topics (not all subscribable).