A Smartphone-controlled Autonomous Robot

Christian Bodenstein*, Michael Tremer *, Jonathan Overhoff†, and Rolf P. Würtz*†
* Institute for Neural Computation, Ruhr-University Bochum, Germany
† Department of Electrical Engineering and Information Technology, Ruhr-University Bochum, Germany

***Abstract*—We present a low cost autonomous robot, which is controlled by a standard android smartphone. We describe the complete mechanic and electronic setup. Stereo vision is derived from the smartphone camera using a mirror system. We further describe a dynamical system controller, which allows the robot to navigate around a room towards a target while avoiding obstacles whose position are estimated by the visual system. For heavier computations, a laptop can be used via wireless LAN. Altogether, this design is useful for a variety of mobile robot experiments.**

***Index Terms*—android software, stereo vision, robot navigation, dynamical system, robot hardware**

I. Introduction

Autonomous robot navigation is one of the classical application areas for natural computing algorithms. There are many unsolved problems, which leave it as a very interesting research field. To make it accessible to many students, robust and cheap hardware is required. Searching for a low price, powerful and robust physical vehicle we did not find anything that fitted our needs. The minimal requirements were that it should be able to navigate an average student flat with, e.g., carpets and door sills by visual control.

Smartphones are omnipresent in our society. We all have them and taken everywhere we go and we also wonder every once in a while how fast they are. They are faster by orders of magnitude than what NASA had when they traveled to the moon, we often hear. For AI applications, smartphones come with a lot of useful features like computational power, cameras and other sensors. In the present study, we built a mobile robot and added some autonomous mobility features.

Beside the hardware platform we created a software platform that is able to recognize three-dimensional obstacles from the built-in camera of the smartphone. That data is used to create a map which is fed into the controller code that steers the robot. The software is available upon request.

This paper starts with the complete information and instructions to build the robot hardware using a standard smartphone for control and vision. Then the software platform is described. Finally we present some of the implemented behaviors.

II. Hardware

The robot hardware includes the chassis, the stereo vision device and the control electronics, which form the interface between chassis and smartphone.

A. Chassis

The chassis is responsible for moving around in the real world. Driving a vehicle to a determined position with a common d.c. voltage motor requires a controlling signal adapted to the dynamic load of the motor. This depends on vehicle mass, friction, velocity, acceleration, terrain inclination, and the electrical impedance of the motor coils. Position feedback of the motor axles is used to optimize the control signal, allowing controlled regulation of the driving speed. Based on this data, the actual position of the vehicle can be estimated by *odometry*, which is notoriously awkward, because inaccuracies accumulate over the entire moving distance. So the main sensor for measuring distances positioning is the camera, but odometry is used as an extra cue. From these requirements we decided to use a caterpillar drive in combination with electrical stepper motors as shown in figure 2.

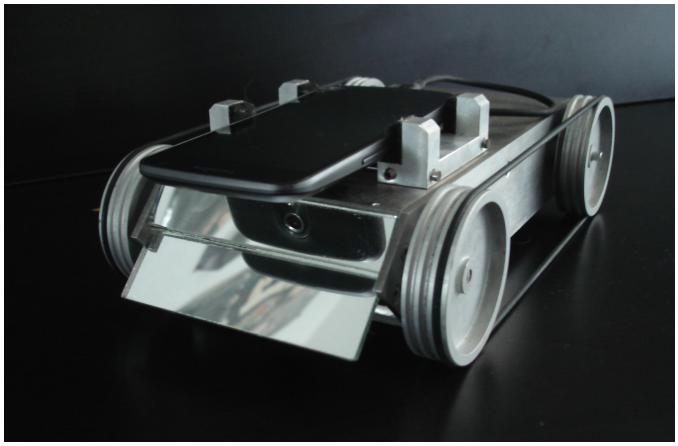


Figure 1. A view of the robot with mounted smartphone and the mirrors enabling stereo vision.

1) Caterpillar Drive: The caterpillar drive allows great contact to the ground at any time, so we can rely much better on the odometry data. It is easy to handle and a robust way to move the robot. A major drawback of this drive is that rotation of the vehicle at slow speeds is difficult, because it depends on a number of conditions like the ground's friction.

2) Stepper Motors: Stepper motors do not need position feedback of the axles for a controlled rotation like d.c. voltage motors do. Instead, one full rotation is divided into a number of equally sized steps. For each control impulse the motor rotates by one step of constant size, what makes odometry easy. Another advantage of stepper motors in comparison to common d.c. voltage motors is the separation of spinning control and torque control. For spinning the motor, an electric current must flow through the coils in a defined sequence. The speed of this sequence regulates the motor speed. The amplitude of the current regulates the motor torque. As a

978-1-4673-7681-5 ©2015 IEEE

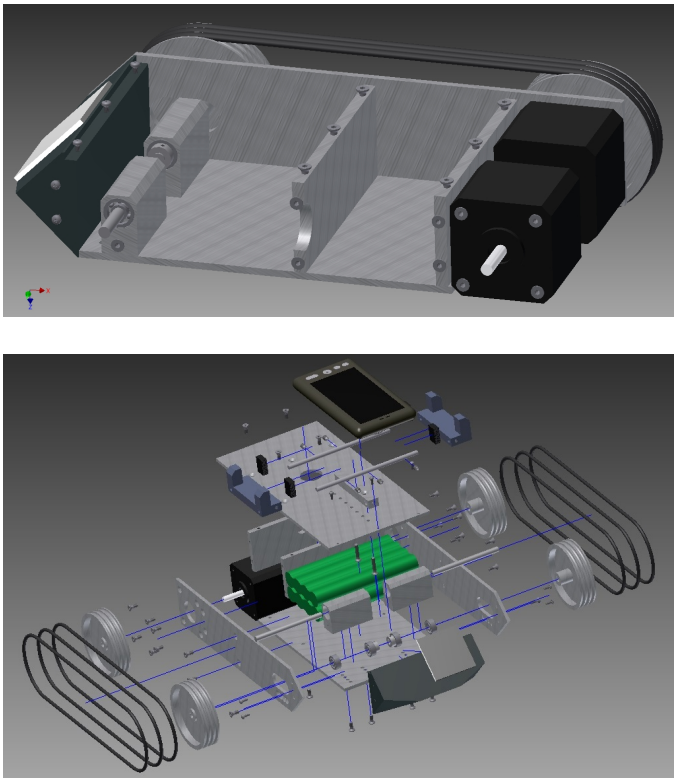
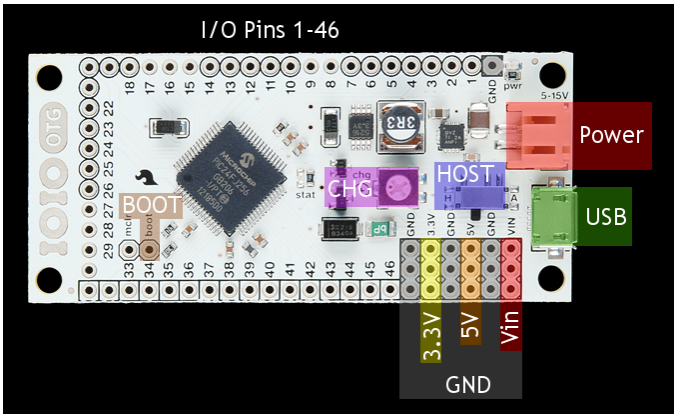


Figure 2. An internal view of the robot chassis and an exploded view of the whole robot without the stereo vision device



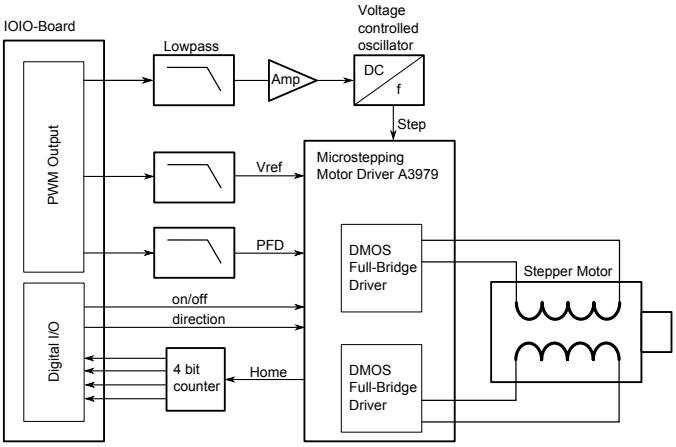


Figure 3. The Android-based *IOIO-OTG* board depicted here extends an Android-based smartphone with 46 physical I/O pins and forms the interface between the smartphone and the electronics, shown in the block diagram below

result, our chassis is very accurate and easy to control. For example when the robot needs to stand still on a slant, the sequence just has to be stopped while the coil current remains at the last sequence position. Now the current amplitude regulates the robot's holding torque.

A disadvantage of stepper motors is that movements tend to be very abrupt. The resulting vibrations hinder the vision device by introducing blur to the images. To smoothen the movement and reduce vibrations we use a concept called *microsteps*. In this concept, the coil currents are stepwise increased as shown on the right in figure 4. We use the microstepping controller *A3979*, which offers sequence generation in full steps, half steps, quarter steps and a sixteenth of a step (compare section II-B2). We picked the 1/16 steps setup, because in this mode the current shape is approximately sinusoidal. This yields maximal smoothness, which is needed for taking sharp camera pictures.

B. Control Electronics

In this subsection we take a closer look at the electronics used to connect the software domain with the chassis. First we shortly present the *IOIO-OTG board*, which acts mainly as an interface between software and hardware. Then we introduce the motor driver *A3979*. After that we describe the peripheral electronics, which are required to make the system work.

1) IOIO-OTG board*:* We use an *IOIO-OTG board* for communication between the smartphone and the chassis control electronics. This interface board can be plugged into an Android smartphone via USB. Then the smartphone can access the 46 physical I/O pins of the board. All pins can be used for 3.3V digital I/O, but most of them have optional functions like 5V digital I/O, analogue input and pulse-width modulation (PWM). For more information, see [1]. In the main operational mode as a host for the smartphone the *IOIO-OTG board* needs an external power supply (5-15V) for operation. The board charges the smartphone and provides a 3.3V/400mA and a 5V/2.5A power supply for external peripherals.

A disadvantage of the board is that the PWM ports can only change the width of the emitted pulses during operation, but not their frequency. Changes in PWM frequency cause intense interruption and generating dynamic frequency signals is not possible. As a result we are not able to create step impulses for the motor drivers via PWM. Instead we use voltage controlled oscillators for step impulse generation, which are controllable via fixed frequency PWMs.

We use the on-board power supplies for powering the motor drivers and a dual 4-bit counter chip. Furthermore, five digital outputs and eight digital inputs are required for controlling and monitoring these devices. Besides the two PWMs for step impulse generation, the board generates two

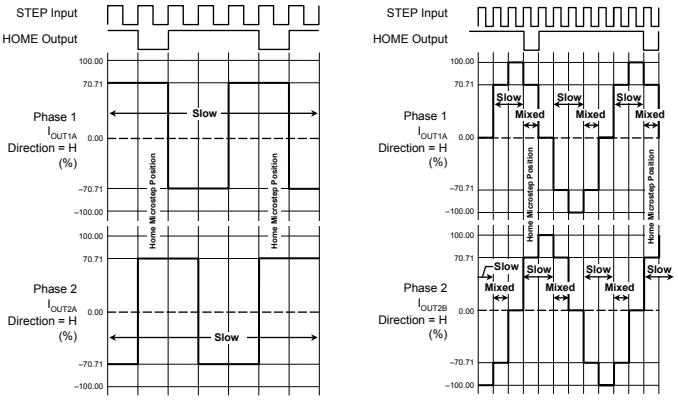


Figure 4. This image shows sequences of the electric currents driven through the coils of the stepper motors (phase 1 and phase 2) by the microstepping controller *A3979*. On the left, the *full-step* mode is shown; on the right the *half-step* mode. The frequency of the incoming step impulses regulates the motor speed; the more intermediate steps are made, the smoother the motor is running. (Source: [2])

PWMs with fixed frequency and adjustable pulse width, which are low-pass-filtered and then used as reference voltages for the motor drivers. Another PWM is used as clock for a charge pump. Five analogous inputs monitor every cell of the lithium polymer battery and the bias voltage of the VCOs. 23 pins remain for optional peripherals.

2) Microstepping Motor Driver A3979*:* The *A3979* is a motor driver for bipolar stepper motors with an output drive capacity of up to 35V and ±2.5A. The driver offers up to 1/16 step microstepping. The *A3979* automatically generates a control signal for the motor coils, based on the following digital and analogue data:

- **step impulses** - Every incoming step impulse rotates the axis of the motor by one (micro-)step. The impulse frequency defines the turning speed.
- **reference voltage** $V_{\mathbf{ref}}$ - The maximum current amplitude (at the top of the approximated sine wave in figure 4) and thus the motor torque is user-defined by this analogous reference voltage. The maximum value should not be greater than $8 \cdot R_S \cdot I_{max}$, where R_S represents the current sensing resistor and $I_{\max}$ the maximum coil current, to prevent the coils from overheating.
- **analogous voltage** $V_{\mathbf{PFD}}$ (percent fast decay voltage). This reference voltage regulates the de-energizing of the motor coils. Thus the user is able to minimize vibrations, which occur, when the shape of the coil current is distorted.

The *A3979* provides a feedback signal called *home*, which consists of an impulse sequence. Each *home* impulse indicates a defined position in the periodic motor control signal and thus a completed full-step. The *home* signal is ideally tailored for odometry. For more information about the *A3979*, see [2].

3) Peripheral electronics: The most basic peripheral device is a power source which powers the whole system. We use a *lithium polymer* battery with a nominal voltage of 11.1V and a capacity of 1000mAh. We monitor each battery cell and shut

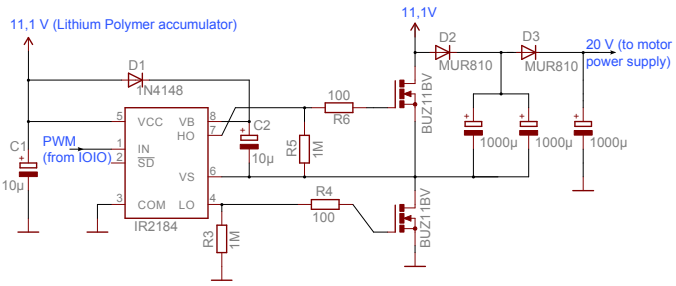


Figure 5. Circuit diagram of the charge pump (circuitry based on [3])

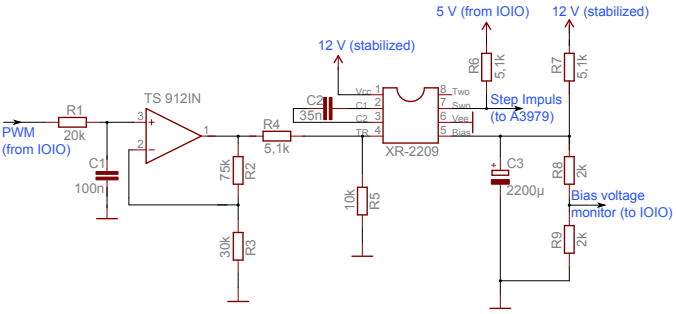


Figure 6. This image shows the circuitry for generating an impulse sequence with variable frequency via a PWM signal with fixed frequency.

the whole system down when a cell voltage drops under a level of 3V.

The motor drivers need almost twice as much voltage as the battery delivers for optimal performance. So we raise the battery voltage with help of a *charge pump* up to approximately 20V (see figure 5). The clock frequency determines the capacity and the ripple voltage of the charge pump and therefore needs to be adjusted to the respective power consumption. We use low-ESR-type (equivalent series resistance) pumping capacitors and have arranged them in parallel to minimize voltage loss and improve heat dissipation.

We generate the step impulse signals with *voltage controlled oscillators* (VCOs). As demonstrated in figure 6, the low-pass-filtered PWM has to be amplified before actuating the VCO circuitry. For this task we to use *rail-to-rail amplifiers* because the voltage level of the control signal can drop to zero. Since the output frequency of the VCOs depends on the supplied voltage, they receive their own *stabilized 12V power supply*.

Caused by our way of generating the step impulse signals, it is not very accurate to do odometry calculations with the input signal data. Instead we use the feedback signals of the motor drivers. But the *IOIO-OTG* board is not able to detect all feedback impulses when the robot drives at a medium speed. So the impulses are counted by two *4-bit counters* which provide an output signal for each bit.

C. Stereo vision device

The main task of our robot is to reconstruct a virtual image of its environment, so it can move around in it. Precisely imaging the physical world in three dimensions requires two (two-dimensional) images of the scene as seen from different

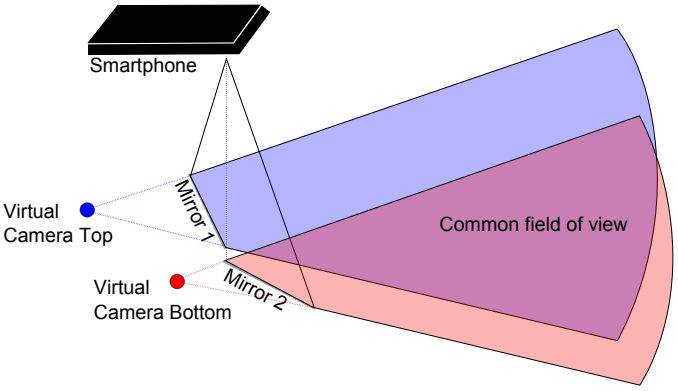


Figure 7. Schematic illustration of the optical system for Stereo Vision with a single camera; the angle and offset between the two mirrors create two virtual camera positions; the distance between these virtual camera positions is fix, so it can be used for three-dimensional reconstruction of the environment.

camera positions. If the distance between the two virtual cameras is known, three-dimensional reconstruction of the scene is feasible. So we needed to build a construction that provided us with the images of two virtual cameras, actually using only the smartphone camera.

Taking two pictures at different positions and measuring the distance with help of odometry is possible, but not optimal. The inaccuracies described above distort the image of the virtual environment even more. So we arranged two mirrors in the way figure 7 demonstrates. The distance between the two virtual camera positions results from the angle and offset between the two mirrors. The horizontal position of the smartphone is very stable when the vehicle is in motion.

D. Smartphones as a brain for robots

The job the smartphone has to do in this project is to control the robot. The robot is able to move itself to where we want to get it, but it has no idea about in what environment it is operating. A brain is needed that can see and understand what it needs to do.

There are plenty of commercial embedded solutions available. They all are basically capable of sending out signals to the hardware which then starts or stops a motor or returns sensory data. That is basically the same what we are doing here, too.

The difference comes with the software we are running here. The Android platform offers loads of things other embedded solutions do not have. Especially in a software project, where rapid prototyping of ideas is necessary, the hardware interfaces should be contained in the robot's software development kit.

Android comes with a very well designed and elaborated programming environment which is well documented and has an API with very rich functionality. The smartphones on which Android is running are fast and cheap. The performance increases by orders of magnitude with every new release and and multiple CPU cores can be found in all of them. Another restriction is the amount of memory available of the phone, which is also not so much of a worry on recent smartphones as it is on other embedded robot platforms.

The powerful community around Android, which provides the world with free libraries for all sorts of tasks, makes the application running on the robot very versatile so that it can do almost any task you can imagine. In the end, we should not forget to mention that smartphones are cheap compared to embedded computers.

For our test case we used a Samsung Galaxy Nexus, which was released in October 2011. It comes with a ARM Cortex-A9 dual-core processor, clocked at 1.2 GHz and 1GB RAM. The integrated camera takes pictures in a resolution of 5 megapixels. Additionally, the phone provides a lot of useful sensors like compass, accelerometer, gyroscope and light sensor.

III. Software

A. Objective

One of the goals of this project was to create a versatile platform for robots, which can easily be enhanced with new functionality. It became quite clear that an embedded solution was not a suitable solution, because the high costs of the hardware and the high amount of hours we had to put into developing some sort of operating system that was able to talk to the hardware and to run the robot.

The search went on with the following in mind:

- The platform should have interfaces for communicating with basic hardware like the camera.
- There should be a user interface, which shows the robot's status in real-time.
- Prototyping ideas should be very easy.
- As we were going to run some heavy computation, there should be enough performance available.

B. Design

Eventually it became clear, that at the current state of the art smartphones were not able to perform all tasks we demanded from them. Lots of modern libraries are able to use the GPU to speed up calculations, but being limited by the power of the used smartphone was not an idea we liked.

Hence we created an architecture that was able to connect to a system with much more power in order to do the most complicated calculations there. In our experiments, we discovered that most of our concerns regarding the inefficiency and lack of performance of the smartphones were false, but due to the design we chose, we can cope with much more complex calculations, even with cheap hardware. Figure 8 shows the distribution of software modules on the smartphone and an optional laptop.

1) App & Client: Dividing the whole application into two parts comes with a long list of benefits. Most importantly, we were able to focus on the task that should be done in the individual parts of the software.

The task of the app, which is the part running on the smartphone, is to control the robot. That is reading and processing sensor data to be used for calculations and also sending control data to the motors. It is realized as an Android

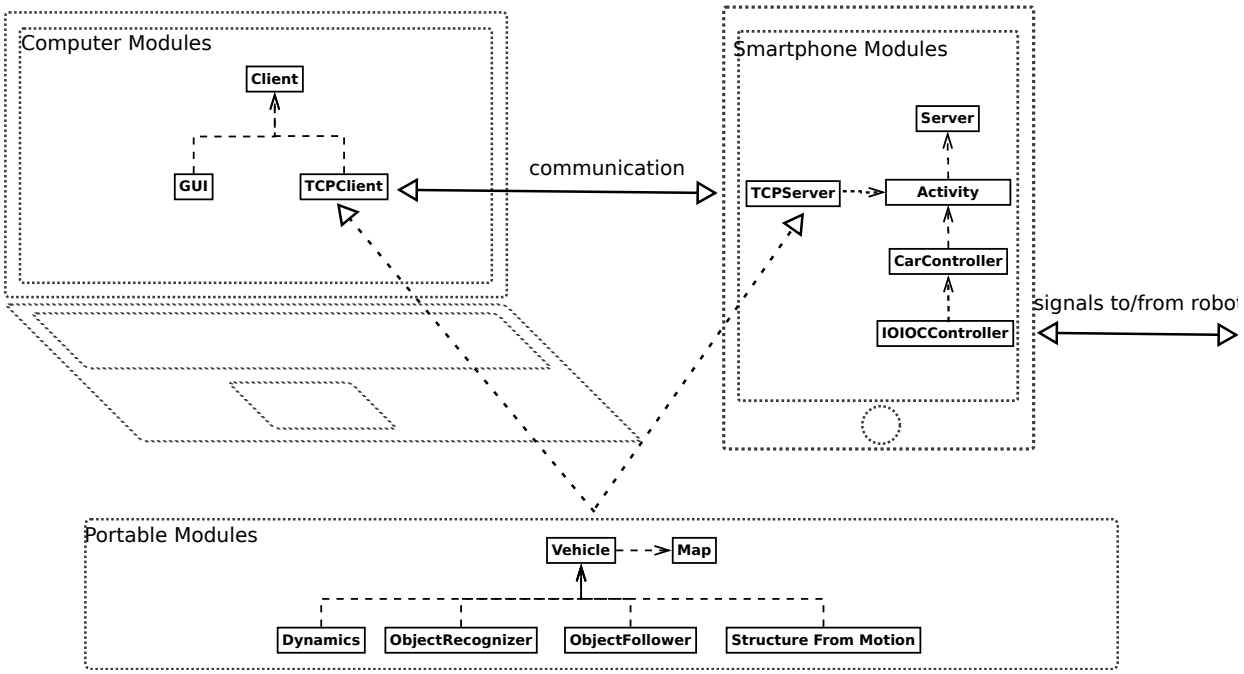


Figure 8. This figure shows the main software modules. Hardware controllers run directly on the smartphone, for interaction a GUI on the laptop is used. Portable modules can run on the smartphone but may also be moved to the laptop if more computational resources are required.

App that is running a server demon, the Client (figure 9) can t to in order to receive the sensor data and send control data.

To be able to move parts of the application from the App to the Client and vice versa, we chose Java as the programming language for both App and Client.

C. Implementation

A modular concept of the application is very important to be able to change things very easily in the future. Therefore we created a class concept where individual classes can be replaced by others, so that other robots can be controlled or the brain can be changed to make the robot behave differently.

1) Class Model:

a) `Vehicle` (Client): The `Vehicle` class is the representation of the physical robot in the software. It implements the most basic operations like sending control data to the robot via its control connection. It is also responsible for getting sensor data and camera images from the robot and stores them, so that the rest of the application can easily access the data and make calculations with it.

This class does not contain any high-level operations like steering the robot to a certain position, but is only able to set the speed of the motors.

b) `CarController` (App): The `CarController` class is the equivalent of `Vehicle` in the App. It receives motor commands from `Vehicle` and sends them to the hardware as well as collecting sensor data.

c) `Map` (Client): All knowledge about the robot's environment is stored in a map. This map is shown in the Client application and used to safely steer the robot around obstacles and towards the target.

Because of the huge size of the map and the small amount of obstacles in the environment `Map` is realized as a linked list, in which each point is represented by a unique index and a probability of how likely it is that there is an obstacle at a certain coordinate in the map. Additionally, the trail of the robot is stored, so that one can trace the path the robot has chosen on the map.

d) Helper Classes: The `Coordinate` class is used to make calculations with geometrical data more easy. It implements functions to compute the distance to an other `Coordinate`, to normalize and compute the length. To get data from the camera of the robot, an extra class which is called `VehicleCamera` has been created to take care of this.

2) Control Connections: For a seamless link between the Client and App software parts, a transparent and fast connection which is able to transport any sort of data is required. For the start this was sensor data which gives us information about the state the robot is in. After that, we required to send control information, so we could instruct the robot to do certain actions like moving somewhere. All modern smartphones come with a wireless LAN interface, which is suitable for this need. There is enough bandwidth available to transport a huge amount of data per second and it has a much better range than other PAN technology like Bluetooth, which also offers much less bandwidth.

Through the wireless link, we created a TCP connection across which serialized Java objects can be transmitted. The App has the role of the server, the Client part of the software is also the TCP client which connects to the server.

To create a proper standard of the messages that

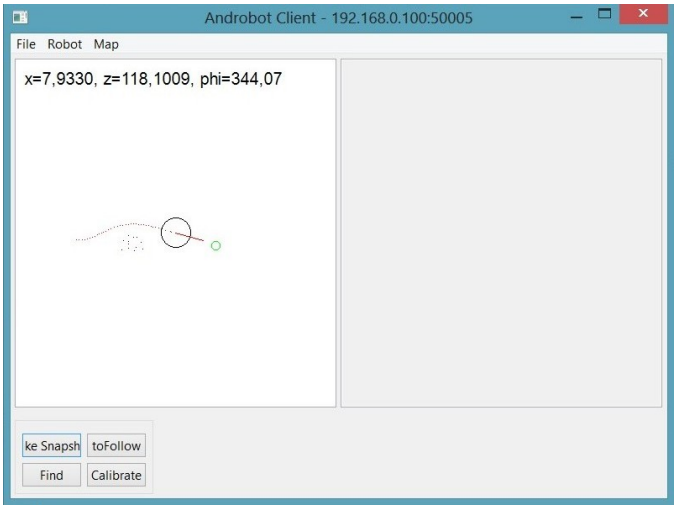


Figure 9. This is the main window of the Client application that is used to show the path and map data of the robot. It is also possible to manually send control commands to the robot to let it move.

are transferred between both communication partners, the `TCPMessage` class has been created. It is able to wrap itself around sensory data and more information and takes care of encoding and decoding all data. For any kind of action the client has to make a request and the server only replies to that with an other `TCPMessage` object. The server is not able to initiate any transfer of information by itself.

a) Video Streaming Connection: A more difficult matter was the fast transfer of camera images in high resolution. This data is not only needed to show a preview in the Client GUI, but all computation is made on the image data as well.

The camera data is encoded in JPEG format, which already provides a high level of compression. The compression is lossy and the more the data is compressed, the more information is dropped from the image. If the compression level is so high that there are compression artifacts which interfere with the analysis of the image even when they not visible to the human eye.

On the other hand, the wireless link is blocked for a long time if an image in high resolution is transferred to the client and no control data can be sent to or sensor data can be requested by the App. To overcome this inconvenience, we added a second TCP connection, which essentially works like the first one, but is dedicated to only transfer image data.

Two different kinds of data are transferred to the client. The first one is image data in high resolution which is used for computation. The metadata of such an image is also very important, because for learning the environment, the software must know where the snapshot was taken. A second kind of image data the Client may request is a much more compressed snapshot from the camera which is only used to render the preview in the Client GUI. There is no additional metadata about the robot's position included.

The two-connection architecture enables us to keep the control connection very responsive, even if we ask for high-resolution image data. The final bottleneck is the wireless connection, for which better hardware can compensate.

IV. Behavior

For the robot to be autonomous rather than just a remotely controlled vehicle, it is necessary to use the provided sensor data and transform them into motor control commands. No other built-in smartphone sensors, e.g. ultrasound sensors, laser distance meters or infra-red sensors but the camera of the Android device have been used.

As image processing has very high computational costs, we decided to use the image stream we already implemented and did the image processing on a laptop computer. In future, it is likely that smartphones will have enough power to perform this calculation themselves. All written code is highly portable, so that it would run on the smartphone as well.

A. Wheel Odometry

For map building and obstacle avoidance it is important to know how far the robot has moved over time. This kind of movement tracking is done by wheel odometry. Every time the robot's wheel is moving by one tick, a counter is increased. From that the robot's movements are calculated as follows:

Let d_l and d_r be the distances driven by the left and right wheel, respectively and r the distance between the left and the right wheel. Now the change of the angle can be calculated by $\Delta\phi = \frac{d_r - d_l}{r}$. $\Delta s = \frac{d_r + d_l}{2}$ is the moved distance since the last measurement.

The change of the x and z coordinates can be calculated by:

$$\Delta z = \begin{cases} d_l \cos(\phi) & \text{if } d_l = d_r \\ \frac{\Delta s}{\Delta \phi} \cos\left(\frac{\pi}{2} + \phi - \Delta\phi\right) + \cos\left(\phi - \frac{\pi}{2}\right) & \text{else} \end{cases} \tag{1}$$

$$\Delta x = \begin{cases} d_l \sin(\phi) & \text{if } d_l = d_r \\ \frac{\Delta s}{\Delta \phi} \sin\left(\frac{\pi}{2} + \phi - \Delta\phi\right) + \sin\left(\phi - \frac{\pi}{2}\right) & \text{else} \end{cases} \tag{2}$$

Wheel odometry is acceptable for short distances. But for longer distances, the errors generated by lost ticks, bad grip and friction will accumulate and the odometry becomes very inaccurate.

B. OpenCV

The Open Computer Vision Library (OpenCV) [4] is a widely used tool for image processing. It provides lots of functions for stereo vision, feature matching, optical flow, face and object recognition, etc. It has traditionally been developed in C++, but newer versions of the library provide wrappers for Java and Android as well. Especially the latter make it a good choice for this project.

1) Feature Matching: Feature matching is the task to find the same groups of pixels in two or more different images. Several feature matching algorithms are available in OpenCV. SURF [5] or SIFT [6] features are commonly used because they are very robust, meaning they often find the corresponding

Figure 10. Visualization of the matched features.

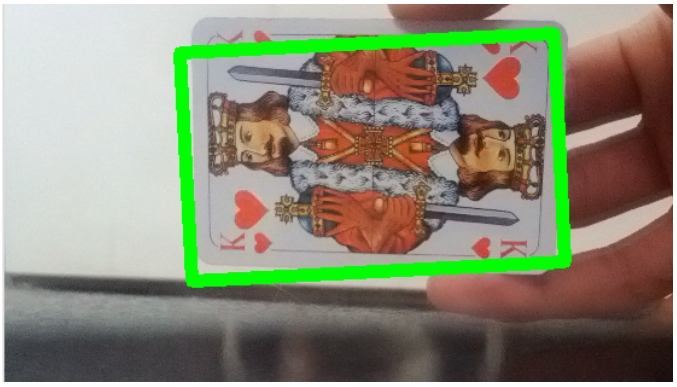


Figure 11. An object found in the image.

features in both images. Their disadvantage is that they are very slow in comparison to other algorithms. ORB [7] is a rotation invariant version of the BRIEF algorithm, which performs much faster than SURF or SIFT. While SURF or SIFT are scale invariant, ORB is not and has a limitation in the number of features that can be found. We decided to pick the best matching method depending on the task we want to do with the found features.

2) Moving towards a given object: The first computer vision example that has been implemented in this project was object recognition. The user shows an object to the robot, and the robot will try to find this object in future images and once it has been found, it will drive towards the object. The algorithm first stores the features of the shown object and matches them with all other future images from the camera. SURF is a good feature matching algorithm for this task, because the features vary much in scale. If enough features are found, the exact position and scale of the object can be calculated.

The robot then uses a simple dynamical system to calculate its forward velocity ($v = (1-s)k_v \frac{\text{ticks}}{\text{s}}$) and rotation velocity ($\dot{\phi} = (c_x - d_x)k_\phi \frac{\text{ticks}}{\text{s}}$) where s is the scale of the object, c_x the x coordinate of the image center and d_x the x position of the object in the image. k_v and k_ϕ are constant. The motors are then simply set to $v + \dot{\phi}$ as the velocity for the left motor and $v - \dot{\phi}$ for the right motor.

3) Stereo Vision: The robot has got two mirrors installed in the front to split the smartphone camera image into a stereo image. With help of stereo vision, the robot is able to build a three-dimensional visualization of the objects in front of it. OpenCV provides many algorithms for stereo vision like stereo camera calibration, feature matching and triangulation.

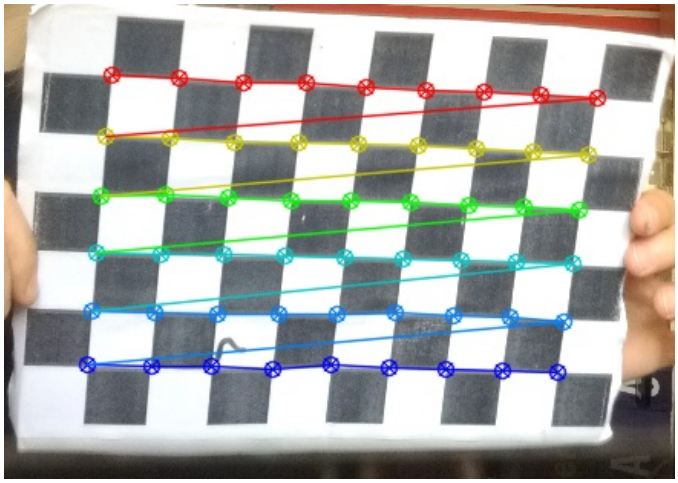


Figure 12. The calibration chessboard.

For calibration, a small chessboard with known pattern size is used. The OpenCV algorithm is able to find this pattern in several images (see figure 12) and estimates the distances between the two virtual cameras. For feature matching, the ORB algorithm is used because of its speed. Wrong features can be easily eliminated on the basis of the epipolar constraint ([8]). The OpenCV function for triangulation uses the rotation and transition matrix of the cameras, which was found in the calibration process and the matched features to find the projection of the points in the three-dimensional space.

4) Structure from Motion: This function is not ready for use, yet, but it does not need any more algorithms than the stereo vision. Structure from Motion is the task to find the movement (rotation and translation) of the camera in two different images. With this information one can threat two images taken over time as stereo pair and triangulate more points. The advantage is that the distance between both camera positions can be greater than the distance between the two virtual cameras generated by the mirrors. This can lead to more accurate 3D points, especially for objects farther away.

The calculated camera movement is also useful to track the motion of the robot. Hence wheel odometry is not very accurate, this method can result in much better trajectories.

C. Movement Dynamics

The user is able to let the robot drive to a certain target coordinate by clicking on the map in the graphical user interface. When approaching the goal, the robot has to avoid obstacles, which can be created by the user by clicking on the map with the right mouse button or they are automatically detected by the robot with help of a stereo vision algorithm. In order to process the large amount of obstacles in the environment (stereo vision can generate several hundreds of them), the robot uses a dynamical system proposed by [9] to calculate the rotation velocity. If where ϕ is the heading direction of the robot and ψ_g is the direction of the target (see figure 13): $\dot{\phi}$ is the change in the heading direction (i.e. rotation velocity) of the robot. With this equation the robot

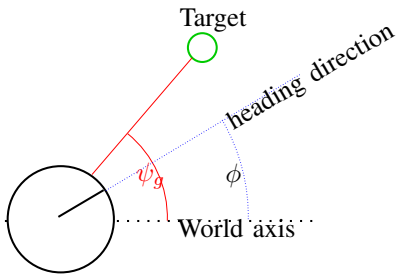


Figure 13. The robot, its heading direction and the angles.

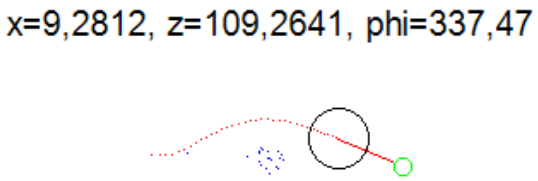


Figure 14. This image shows the internal representation of the robot's trajectory (red dots at equal time intervals), the identified target (green) and the found obstacle points (blue), once the target was reached.

will always turn towards the target.

$$\dot{\phi} = -(\psi_g - \phi)\,. \tag{3}$$

The equation

$$\dot{\phi} = (\psi_g - \phi) \tag{4}$$

makes the robot avoid the target. A visualization is shown in figure 14. We slightly modified this equation to generate the following dynamic system

$$\dot{\phi_g} = -k_g \sin(\psi_g - \phi)(e^{-c_1 d_g} + c_2)\,, \tag{5}$$

where k_g, c_1 and c_2 are constants and d_g is the Euclidean distance to the target. The angle between heading direction and target $(\psi_g - \phi)$ is surrounded by a sine function to generate a smooth turning rate, even if the angle jumps between 0° and 360°, which are in fact the same angle. The smaller the distance to the target d_g, the greater is the turning rate, so that the goal won't be missed. For this, d_g is passed into a negative exponential equation. The constant c_1 is used to adjust the linearity of this equation and c_2 is used to create a minimal rotation velocity towards the target, even while far away.

The obstacles are represented as hundreds of individual points found by the stereo vision algorithm. With the help of the sum of dynamical equations, the robot can calculate a 'mean' obstacle that it tries to avoid as computed by

$$\dot{\phi}_o = k_o \underbrace{\frac{1}{N+\tau}}_{\text{Normalization}} \sum_{i=0}^{N} \Big(\underbrace{\text{sign}(\psi_{o_i} - \phi) e^{-(\psi_{o_i} - \phi)^2}}_{\text{Gaussian dynamics}} \underbrace{e^{-d_{o_i} c_3}}_{\text{Distance term}} \Big) \tag{6}$$

with the following parameters:

- k_o: constant
- N: number of obstacle points
- τ: threshold value. If few obstacle points were found, the function output will be reduced.
- ψ_{o_i}: the angle to the i'th obstacle point.
- ϕ: the heading direction of the robot.
- d_{o_i}: the distance to the i'th obstacle point.
- c_3: constant.

The rotation velocity will be calculated by $\dot{\phi} = \dot{\phi}_o + \dot{\phi}_g$. With help of these dynamics, the robot is able to avoid complex configurations of obstacles on its way towards the target without hitting any obstacle coordinates.

V. Conclusion

We have constructed an autonomous robot from readily available hardware and a smartphone, implemented a control infrastructure and some behavior components. The mirror-aided stereo vision system offers much potential to let the robot view the environment that surrounds it. Our tests show that even with low resolution images close objects can be reconstructed. With higher resolution images and with help of the structure from motion approach, the robot is be able to map its environment.

The IOIO board makes it easy to add more components like sensors or actuators (e.g., a small robot arm) to the robot. The constantly increasing performance of the newest smartphone generates, will enable more complex behaviors in the future. Speech recognition (which has already been implemented in newer Android releases) and the ability to be connected to the Internet all the time are significant features of the system that may become more important over time. For example, imagine an autonomous robot cloud where each robot can upload items that it has learned to share them with other robotics.

We are confident that a system with the features and advantages presented here can help to build cost efficient robots, which are used by scientists to further the progress of autonomous robots.

References

[1] : IOIO-OGT documentation https://www.sparkfun.com/tutorials/ 280.
[2] Allegro MicroSystems, I.: Allegro A3979 - Microstepping DMOS Driver with Translator – Datasheet
[3] Teuteberg, H.D.: 22V 5 Ampere aus 12V http://www.atx-netzteil.de/22v_5_ampere_aus_12v.html.
[4] : OpenCV http://opencv.org/.
[5] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (surf). Computer vision and image understanding **110**(3) (2008) 346–359
[6] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60**(2) (2004) 91–110
[7] Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: an efficient alternative to SIFT or SURF. In: Proc. ICCV, IEEE (2011) 2564–2571
[8] Xu, G., Zhang, Z.: Epipolar Geometry in Stereo, Motion and Object Recognition: A Unified Approach. Springer (1996)
[9] Schöner, G., Dose, M., Engels, C.: Dynamics of behavior: Theory and applications for autonomous robot architectures. Robotics and Autonomous Systems **16**(2) (1995) 213–245