B.Sc. Thesis

Autonomous Car Driving using a Low-Cost On-Board Computer

Intelligent Autonomous Systems Lab

Brian Pfretzschner

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Studienarbeit ohne fremde Hilfe und nur unter Verwendung der zulässigen Mittel sowie der angegebenen Literatur angefertigt habe. Mir ist bekannt, dass die Weitergabe von Rechten an dieser Arbeit oder von Auszügen aus dieser Arbeit an Dritte der Zustimmung von Herr Jan Peters bedarf.

Darmstadt, July 12, 2013

(B. Pfretzschner)

	Technische Universitaet Darmstadt Department of Computer Science Intelligent Autonomous Systems Lab
Advisor:	Marc Deisenroth
1st Referee:	Marc Deisenroth
2nd Referee:	Jan Peters

AUTONOMOUS CAR DRIVING USING A LOW-COST ON-BOARD COMPUTER

Submitted by:	Brian Pfretzschner
Matriculation number:	1572582
Degree course:	Informatics
Filing date:	June 30, 2013

Contents

List of Figures

List of Tables

Listings

1 Abstract

Autonomous car driving is a sophisticated, interdisciplinary topic that consists of various partially independent tasks like recognition of the environment, movement control or decision making. It is generally an interesting topic and is already under research with cars in all sizes. There is also a challenge[1] to compare various attempts on autonomous passenger cars. In this thesis, we are using a low-cost RC car in combination with a modern microcomputer to build an autonomous low-cost car. In contrast to prior work, we do all computation on-board, including recognition of the environment and decision making. This is possible because of new more powerful hardware and novel algorithms for image processing. The result is that we did not even use to the full, so there is still spare capacity for extensions or more complicated tasks, for instance to control a more complex and powerful car in a bigger, catchier environment.

[1] The *Darpa Grand Challenge* is an international competition to compare driverless cars operating on realistic long distance routes.

2 Introduction

Autonomous car driving is the attempt to operate a car intelligent within its environment, controlled by an on-board or off-board computer. The computer is supposed to navigate at its best through predefined areas like streets or marked-out routes. To achieve that, sensors like cameras or radar are required to recognise the environment.

The objective of this thesis is to build such an autonomous car based on a standard low-cost RC car and replace the original remote control unit by a microcomputer. To perceive the the environment, the microcomputer will have a webcam attached. The task of this microcomputer is to recognise a target object and learn how to control the car to drive to this object as visualized in figure 2.1. Consequently there a two main tasks. First, the microcomputer needs to detect the object and process its position. Second, it has to learn how to control the car considering its current position. The microcomputer shall be able to operate autonomously especially without a powerful computer connected that does the tedious tasks.

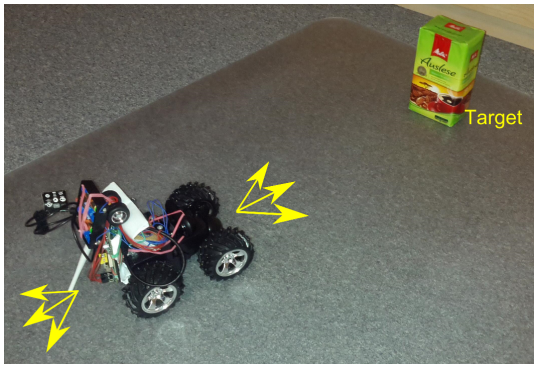


Figure 2.1.: Scenario.

The microcomputer that is used is a Raspberry Pi, see figure 3.4. It is a modern credit-card sized computer, that is very powerful given its size and price[1]. Of course it cannot keep pace with a modern desktop PC but they are hard to compare since they have different fields of application. The Raspberry Pi's advantages are its small size and the low power consumption. Therefore, it is a promising platform for mobile on-board computing.

In contrast to a normal desktop PC, it has 26 general purpose input/output pins (GPIO). Eight do not have a preallocated purpose but 18 of them are intended for special functions like bus connectors or a PWM[2] signal. Nevertheless, they can be used for any purpose as well. Because of those features, the Raspberry Pi is well suited for many variable applications.

There was such a high demand after the initial release of the Raspberry Pi that there was a *one-per-customer* restriction which has been cancelled in July 2012[3].

[1] The Raspberry Pi Foundation compares the Raspberry Pi with a Pentium 2 PC with 300MHz (`www.raspberrypi.org/faqs`).

[2] Pulse-width modulation (PWM) is a rectangular pulse wave which can, for instance, be used as clock signal.

[3] `www.raspberrypi.org/archives/1588`

3 Hardware

The hardware is composed of three parts:

1. Remote-controlled car,
2. Raspberry Pi (including webcam and WLAN adapter),
3. Interlayer.

All components between Raspberry Pi and the car are called *interlayer* by now. The car as well as the Raspberry Pi have only connections to the interlayer (see figure 3.1), which transforms and converts the signals between both of them.

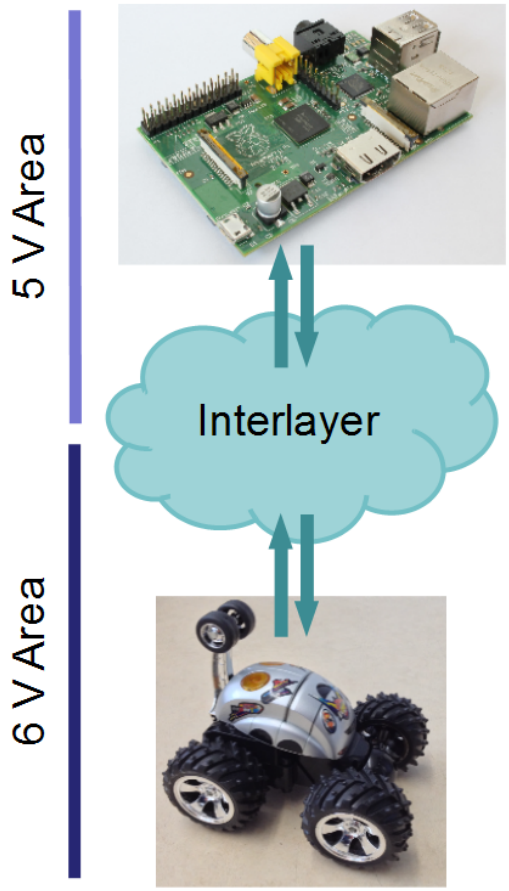


Figure 3.1.: Overview of the hardware elements and their relations.

Compared with an animal the car is like the skeleton and muscles, the interlayer would be the nervous system and the Raspberry Pi the brain.

3.1 Car

The car is a market-based toy. It is called Panther Chariot and is probably produced by MATARO[1]. It has a non-standard driving behaviour because the front wheels do not work as usual. Instead, the entire front axle rotates like a propeller of a plane (see figure 3.2).

Figure 3.2.: An illustration of the non-standard driving behaviour.

The car offers two motors and a slot for the batteries. The rear motor is used to drive the rear axle and, thus, move the car forwards and backwards. The second one is the front motor, which rotates the entire front axle as visualized in figure 3.2. The front wheels themselves are not driven.

The battery slot is applicable to hold four AA batteries. Each battery provides 1.5V, so all together the internal battery case provides 6V.

Motors

The motors are standard low cost motors. The front motor has a transmission build into the case, to support the motor. The transmission is required since the front motor has to raise the entire car when starting the rotation. Therefore, it needs a high torque and a low rotational speed. Unfortunately, the motor is not specialized for this use case and has therefore medium torque and rotational speed.

The rear motor has also a transmission but this gearbox is directly attached to the motor (see figure 3.3). The rear motor does not require a high torque like the front motor does, because it just pushes the car either direction. That is probably the reason why the gearbox is smaller and directly attached to the motor.

The electric current flow through a motor can be calculated based on the resistance and the voltage. The voltage is known to be 6V and a measurement of the resistance of the front motor resulted in about 2.3Ω. Using those values, the formula tell us that the front motor has a peak current of

[1] www.mataro.co.uk/shop/Panther_chariot_radio_controlled_car.htm

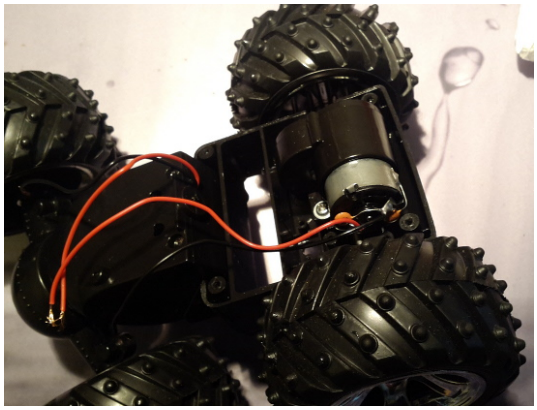


Figure 3.3.: Below view of the RC car with uncovered rear motor.

$I = \frac{U}{R} = \frac{6V}{2.3\Omega} \approx 2A$. Using this current, the electric power the front motor can take at the maximum is $P = U * I \approx 6V * 2A \approx 12W$.

3.2 Raspberry Pi

As mentioned before, the Raspberry Pi is a modern credit-card sized computer as shown in figure 3.4. It is comparable in computational power and memory size to a desktop PC 15 years back. It has similar capabilities but is at the same time much smaller and requires less power.

Model B was released in February 2012 before the lower-cost model A was released in February 2013, because the expected demand for model A was lower. Model B (revision 2) is the main model which is more powerful than model A (see table A.1). Both models use the same CPU and GPU, whereby the CPU uses the ARM architecture. Therefore, the operating system and every program that should run on the Raspberry Pi has to be compiled to the ARM architecture. The advantage of ARM processors in comparison to x86 or x86_64 processors is that the ARM instruction set is designed as RISC[2], which means there are less commands but these are executed really fast. In contrast, x86 or x86_64 processors use a CISC[3] instruction set, which provides more and more feature-rich commands but they also need more time for execution. Additionally, CISC processors internally consist of more transistors since they need to handle more different commands than RISC processors. Fewer transistors mean less power consumption and heat and, of course, also result in a lower price.

Both models use the same CPU and GPU. The CPU has 700Mhz and can be overclocked up to 1GHz without losing the warranty. The main purpose of the GPU is to enable the Raspberry Pi to play 1080p[4] videos smoothly and is therefore not used in context of this thesis.

The main difference between both models is the main memory capacity and number of available ports. Model B is better equipped with 512MB of RAM and, among others, two USB 2.0 and one Ethernet port. Model A has only 256MB of RAM, one USB 2.0 port but no Ethernet. In return model A has a lower power consumption than model B. Model B is available in two versions, so called revisions (rev). Rev 1 differs from rev 2 just by its RAM size since it has only 256MB instead of 512MB. Thus, model B rev 1 is comparable with the model A with an additional USB port and Ethernet but also with a higher energy

[2] Reduced instruction set computing (RISC).
[3] Complex instruction set computing (CISC).
[4] 1080p is a high-definition video standard with a resolution of 1920x1080 pixels also known as Full HD.

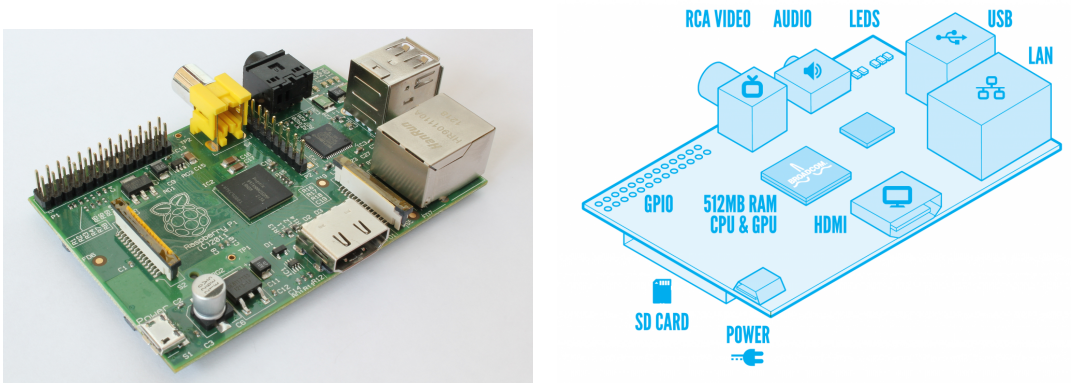


Figure 3.4.: Raspberry Pi, model B, revision 2.

Source: Paul Beech, Raspberry Pi Foundation, `www.raspberrypi.org`.

consumption and price. To conclude the advantage of model B rev 2 is to have more main memory and ports. Model A in contrast requires less energy. See table A.1 for detailed information and table 3.1 for an illustration of the differences between the models and revisions.

	Model A	**Model B**	
		Rev 1	Rev 2
Ports:	−	+	+
RAM:	−	−	+
Power:	+	−	−
Price:	+	−	−

Table 3.1.: Raspberry Pi models and revisions comparison. ("+" means better, "−" means worse.)

3.2.1 Webcam

The Raspberry Pi needs to determine the current position of the car relative to a target object, in this context represented by a coffee box. The easiest way to achieve that is using an ordinary webcam. There are other more sophisticated and more pretentious ways, for instance, the use of a Microsoft Kinect[5] camera instead. This would bring some advantages compared with an ordinary webcam, like depth sensing. At the same time, a Kinect is expensive and does require more power.

The webcam that is used in context of this thesis is a *Logitech QuickCam Pro* (see figure 3.5(a)). It is a modern webcam intended to be used for private purposes. In the context of this thesis, it has some benefits like a good resolution of 2 Megapixels which results in good images. Additionally, it has an autofocus. This is really important since the car is supposed to be in different positions relative to the target object. If the camera could not automatically focus the object, the detection range would be a lot

[5] A special motion sensing camera initially created for a video game console which has some additional features like depth sensing.

(a) Webcam. (b) WLAN adapter.

Figure 3.5.: Webcam and wireless adapter.

smaller. Another simple advantage is the small construction form and the short cable. This is helpful because the camera has to be mounted on the car without interfering the cars moving parts.

3.2.2 Wireless LAN

A wireless adapter in not mandatory for the task of this thesis. Nevertheless, it makes life a lot easier because it enables us to be connected to the Raspberry Pi while it is operating. Thereby, it is possible to get live debugging messages and see what is going on while the car is learning.

Since there are no special requirements to the wireless adapter, it is preferable that it works without high configuration effort. The *TP-Link TL-WN722N USB* (see figure 3.5(b)) is a wireless adapter which is known to work well with the Raspbian operating system (based on Debian Linux, see section 3.2.3). The adapter is huge in comparison to other wireless adapters but uses an external high gain antenna which improves signal quality and connection stability.

More about the wireless network and its configuration in section 3.2.3.

3.2.3 Operating System

The Raspberry Pi has no classical hard drive like a desktop PC. Instead it uses a normal SD card for this purpose. Since the Raspberry Pi is delivered without a SD card, it has no operation system (OS) installed by default. Nevertheless, there are optimised versions of established Linux distributions such as Debian[6] or Arch Linux[7] available. The modified Debian is called called *Raspbian*.

Debian sees itself as “The Universal Operating System”. In fact, it is very adaptable and used in different environments for example on desktop PCs and servers as well as on embedded devices such as NAS (network attached storage).

[6] `www.debian.org`
[7] `www.archlinux.org`

Raspbian is explicitly recommended at the official Raspberry Pi website[8]. That is the main reason why it is used in this context. The current Raspbian (at the time of the installation of the operating system) was based on Debian *wheezy*. Wheezy is the official name for Debian 7.0, which is still under development. One of the new features in Debian wheezy is its official support of *armhf*. This stands for ARM architecture with *hard-float* support.

Hard-float EABI

An embedded application binary interface (EABI) specifies how a software program should *work* on a specific hardware architecture. This is done by defining how data types should look like, how specific functions calls should be done and so on. It also defines hardware characteristics and abilities.

armhf means ARM architecture with *hard-float*. The opposite of hard-float is soft-float which handles all floating point operations in software[9]. Hard-float, by contrast, delegates those operations to the Floating Point Unit (FPU), which is a lot faster. Of course this requires an FPU, which is not invariably the case. The Raspberry Pi is equipped with an FPU so it is reasonable to make use of it.

Installation

The installation is straightforward. We only have to download the archive of the operating system and extract it. In there is a *.img* file that is a exact copy of a drive. The only thing that needs to be done is to write this image to the SD card. With Linux this can be achieved by using the program dd as shown here:

```
1 $ dd if=2013-02-09-wheezy-raspbian.img of=/dev/sdc
```

`if` (input file) sets the path to the image file that should be written to `of` (output file), which is in this case `/dev/sdc`. An SD card with at least 4GB of space is advisable. In context of this thesis a 8GB SD card is used. Unfortunately, the SD card offers only 7.1GB of space. Right after the installation, the operating system already takes about 1.4GB.

Since Raspbian includes a graphical interface, which is not required for this thesis, it makes sense to remove those features to free SD card space and decrease the time required to boot. The following commands also remove some other not required packages and freed about 350MB of SD card space.

```
1 apt-get purge x11-common libgtk* python
2 apt-get autoremove
```

The fresh booted and idle system takes only about 60MB of main memory. This is mainly used by the Linux kernel, but there are also some daemons[10] running. In addition, some memory is required for the SSH[11] connection, which is used to read those values. The current memory consumption as well as the total available memory can be retrieved using `cat /proc/meminfo`:

[8] `www.raspberrypi.org/downloads`
[9] The operations are simulated with multiple integer operations.
[10] A Unix daemon is a process that runs in background, for example sshd, which is responsible for handling incoming SSH connections.
[11] Secure Shell is one way to login into a remote system.

```
1 $ cat /proc/meminfo
2 MemTotal:         497544 kB
3 MemFree:          439152 kB
4 [...]
```

The command `cat /proc/cpuinfo` outputs some information about the Raspberry Pi:

```
1 $ cat /proc/cpuinfo
2 Processor       : ARMv6-compatible processor rev 7 (v6l)
3 BogoMIPS        : 464.48
4 Features        : swp half thumb fastmult vfp edsp java tls
5 [...]
6 Hardware        : BCM2708
7 Revision        : 000e
8 Serial          : 0000000084bf4902
```

An important value is `Revision` in line 7. The Raspberry Pi that is used in this thesis has the value `000e` which means it is a model B, rev 2. Another important value is `vfp` below `Features` in line 4. It proves that the processor has a floating point unit attached.

WLAN

The only device that needs to be configured is the wireless network. The interface is detected automatically but some additional information is required to automatically connect to a local wireless network.

Therefore, we need to edit the file `/etc/network/interfaces`. This is the main network configuration file in Debian. To automatically enable the wireless interface and connect to the local wireless network, the following lines have to be added to the *interfaces* file:

```
1 auto wlan0
2 iface wlan0 inet dhcp
3     wpa-ssid <network-name>
4     wpa-psk <network-password>
```

`wlan0` is the name of the wireless adapter. Whether it should use DHCP or a static IP address is specified in line 2. Finally, the SSID of the network and its password are required on lines 3 and 4. This configuration assumes that the wireless network uses the encryption method WPA or WPA2.

To test the new configuration you can restart the Raspberry Pi‘s network by executing:

```
1 /etc/init.d/networking restart
```

Afterwards, the connection should be established successfully. If so, as from now the Raspberry Pi will connect automatically to this wireless network while booting.

3.3 Interlayer

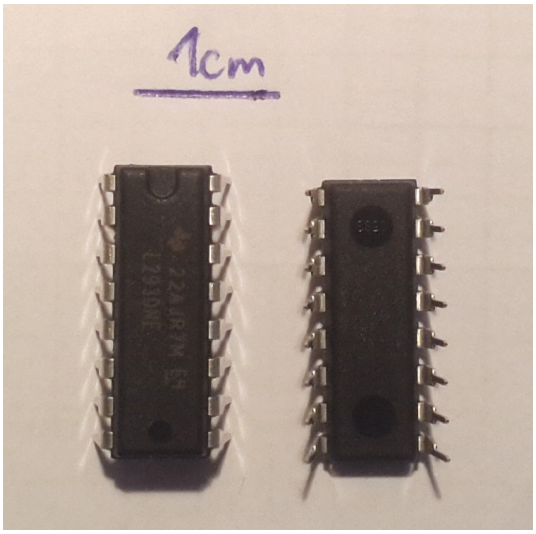
The interlayer are the components that are required to connect the car with the Raspberry Pi. Consequently, the interlayer has two main purposes: Motor control and power supply for the Raspberry Pi.

Motor control means that the high current and voltage signals which are used to power the motors have to be switched by the low current and voltage signals from the Raspberry Pi. As the second task, the interlayer has to handle the power supply for the Raspberry Pi since the internal batteries supply 6V but the Raspberry Pi requires 5V.

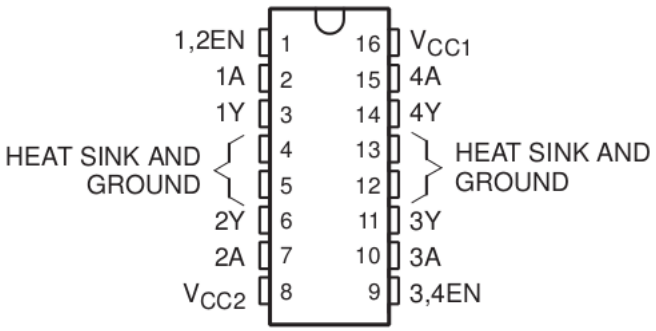
3.3.1 Motor Control

The GPIO signals of the Raspberry Pi provide 3.3V with a low current of about 10mA. Since a GPIO port is not supposed to supply the power for a device, the motors cannot be connected directly to them. For that reason, a specialized component is required to switch the higher voltage of 6V and current of up to 2A. The Integrated Circuit (IC) L293D is well suited for that case (see figure 3.6) since it is supposed to control two motors bidirectionally or four motors monodirectional. Additionally, it has some useful features integrated:

- Separate logic input to minimize IC voltage dissipation.
- Thermal shut down at overuse.
- Output diodes to suppress inductive currents caused by the motors.



(a) Real image of the L293D.



(b) Schematic image of the L293D (by Texas Instruments datasheet).

Figure 3.6.: The integrated circuit L293D used for motor control.

Unfortunately the L293D has also a *small* disadvantage. The motors can take up to 2A as mentioned above, but the L293D can only output a peak current of 1.2A and a permanent current of 0.6A. This causes the motors to run weaker as they technically could.

Motor control in software

The motors are controlled by changing the values of the regarding GPIOs. This can be achieved by simple file operations on a virtual file system provided by the Raspbian operating system. This file system is called `sysfs`, because it is internally connected with the Linux kernel. Any operation on this file system is synonymous to an interaction with the Linux kernel. This is similar to a `syscall`[12] but easier to use.

Those operations can also be invoked in a normal console using standard Linux commands as shown in listing 3.1). To do so, it is important to be logged in as root since those commands need privileged rights. This is also the reason why the final program has to run as root.

Listing 3.1: Manipulation of the GPIO values using a Linux console.

```
1 $ echo "17"  > /sys/class/gpio/export
2 $ echo "out" > /sys/class/gpio/gpio17/direction
3 $ echo "1"   > /sys/class/gpio/gpio17/value
```

The command in line 1 writes the value `17` to the `gpio/export` file where 17 is the number of the GPIO that should be exported to the *user space*. This is required because the Linux kernel separates such components into the *kernel space* and *user space*. Any GPIO is initially owned by the kernel space, so the Linux kernel controls them. The export process brings the GPIO into the user space by moving the control ability from kernel spaces to user space. Thereafter, any program can use the exported GPIO, if it has the corresponding permissions. When the export was successful, a new folder, in this case `gpio17`, was created. Within this folder, there are some files to change the properties for this single GPIO pin. In line 2, the value `out` is written to one of those newly created files called `direction`. There are two possible values that can be written into this file: `in` and `out`. `in` would define the GPIO as an input pin, which means it would only be possible to read whatever the signal on the GPIO pin is high (+3.3V) or low (GND). This setting is for instance required if the GPIO is supposed to read the values of a button. Since no inputs are required in the context of this thesis, we only use the value `out`, which sets the GPIO into output mode. Subsequently, it is possible to specify the value of the pin as shown in line 3. There are two possible values that can be written to the `value` file within the GPIO folder: `0` and `1`. `0` would drive the pin to electrical GND, which means low and `1` represents high, which would bring it to +3.3V.

Using these GPIO pins, the L293D can be instructed to control the motors. It has two direction inputs for each motor, those are `1A` and `2A` or `4A` and `3A`. In table 3.2 is visualized which inputs result in which outputs and thereby motor behaviours. If both inputs have the equal value, both corresponding outputs are equals as well and, for this reason, the connected motor is switched off. If the GPIO values are not equal, the motor is turned on. In which direction the motor is running is determined by the way the GPIO values differ.

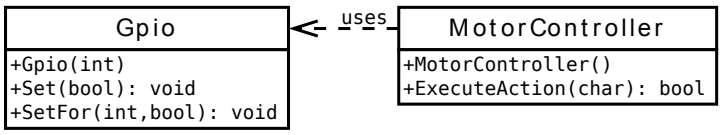
[12] A `syscall` is how a program communicates with the operating system.

Input 1	Input 2	Output 1	Output 2	Resulting motor behaviour
0	0	GND	GND	off
0	1	GND	V_{CC2}	on in direction 1
1	0	V_{CC2}	GND	on in direction 2
1	1	V_{CC2}	V_{CC2}	off

Table 3.2.: Comparison between inputs and outputs of the L293D. `0` means GND or 0V and `1` means +3.3V.

Implementation

In code, the motor control is split into two domains. The first one is about setting GPIO values and the second one handles actual movement commands.



The first class is called `GPIO`. This class is supposed to initialize a GPIO and control its value. It does exactly what is described above in section *Motor control in software*. The constructor of this class expects the desired GPIO number. At the time an instance of this class is created, the GPIO port will be initialized and set to `0`. Afterwards, it is possible to specify the value for this GPIO by using the `Set(bool)` command and pass the new value in. Additionally, there is a function `SetFor(int,bool)` that can be used to change the value for a period of time. After the given time in milliseconds expires, the value is reset to the value before calling this function.

The second class `MotorController` composes `GPIO` classes to create actual movements. When a new `MotorController` instance is created, the constructor will create all required `GPIO` instances. By then, the function `ExecuteAction(char)` can be used to perform predefined movements. All available movements are shown in table 3.3.

There are four elemental actions, since there are two motors that can run in two directions each. These actions are "drive forward" or "drive backward" and "rotate the front axle left" or "rotate the front axle right". There are other actions possible because an action can be aggregated of elemental actions. Elemental actions do only control one GPIO that is turned on, and after a timespan turned off again. By contrast, aggregated actions do control more than one GPIO and change the value of those GPIOs potentially multiple times as shown in table 3.3. They are supposed to perform a more complex movement. There are four additional actions, and one special action. The additional actions are "go forward" or "go backward" *and* at the same time "turn left" or "turn right". This cannot be done just by starting both motors simultaneously in one direction, because the rear motor would push the car although the front axle did not move yet due to the fact that the front motor needs a lot more time to perform its action than the rear motor. Besides of that, both motors must share the available energy and the shortened energy is not enough for the front motor to operate because it has to lift the entire

Movement identifier	Effect	Control sequence
`f`	Go just forward	Rear motor on for 250ms (direction forward)
`b`	Go just backward	Rear motor on for 250ms (direction backward)
`l`	Rotate just left	Front motor on for 500ms (direction left)
`r`	Rotate just right	Front motor on for 500ms (direction right)
`L/R`	Go to the front left/right	1.) Front motor on (direction left/right) 2.) Wait 250ms 3.) Rear motor on for 250ms (direction forward) 4.) Front motor off 5.) Rear motor on for 125ms (direction forward)
`K/E`	Go to the back left/right	1.) Front motor on (direction left/right) 2.) Wait 250ms 3.) Rear motor on for 250ms (direction backward) 4.) Front motor off 5.) Rear motor on for 125ms (direction backward)
`0`	Do nothing	

Table 3.3.: List of available motor control commands.

car. Therefore, the front motor has to start a little bit earlier, so it has a small advance. After the front axle rotated a little, the rear motor is started to push the car in the specified direction. The result is a movement that is similar to steering.

Unfortunately, the current angle of the front axle is unknown and cannot be measured by the hardware that is used in the context of this thesis. This is a problem because, especially the front motor does not always behave identically. In a unpredictable manner, it sometimes works very well, and in the next second it is not strong enough to achieve a rotation any more. Besides that, the time needed to perform a total rotation differs strongly. Also, the front motor works better if it is used recently in comparison to a cold start. That results in different outcomes after an action was executed. In the best case, all four wheels touch the ground again. Sad to say, but often the front axle stays bent after an action including the front motor was executed. This is an issue, the controlling software has to deal with, since there is no feedback that could be used to determine the current situation.

3.3.2 Power supply

The second job of the interlayer is to provide a stable power supply for the motors and most of all for the Raspberry Pi. The power source is the internal battery case that holds four AA batteries. Since each battery provides 1.5V, a total of 6V is available.

The Raspberry Pi normally gathers its power by its USB port. The USB standard defines a power supply voltage of 5V (± 0.25V). Unfortunately, the internal batteries provide 6V. Therefore, it is necessary to regulate the voltage within the interlayer.

There is also another way to power the Raspberry Pi using its static power pins as power input. This method is not recommend because the USB port has circuit protection components attached that the static power pins does not have.

There a various different ways how the voltage can be regulated. The following list is not exhausting but contains some possible methods and an explanation why this method would or would not work in this context.

Standard Voltage Regulator 7805

A voltage regulator is a basic component that maintains a certain output voltage. These components are very small and cheap. A 7805 is a standard fixed-output voltage regulator with an output voltage of 5V. Unfortunately, most standard voltage regulators need an input voltage that is at least 2V higher than the output voltage, which in this case means at least 7V (see A.2). Therefore, the 7805 voltage regulator cannot be used with an input voltage of 6V.

DC-DC Converter

DC-DC converters have a higher efficiency than voltage regulators. This means, while a voltage regulator simply wastes excessive energy as heat, the converter does not absorb this energy in the first place. A DC-DC converter is huge and expensive in contrast to a voltage regulator. Therefore, it is not used in this context, mainly due to its price. Additionally, the size would also be a problem, since the available space on the car is limited.

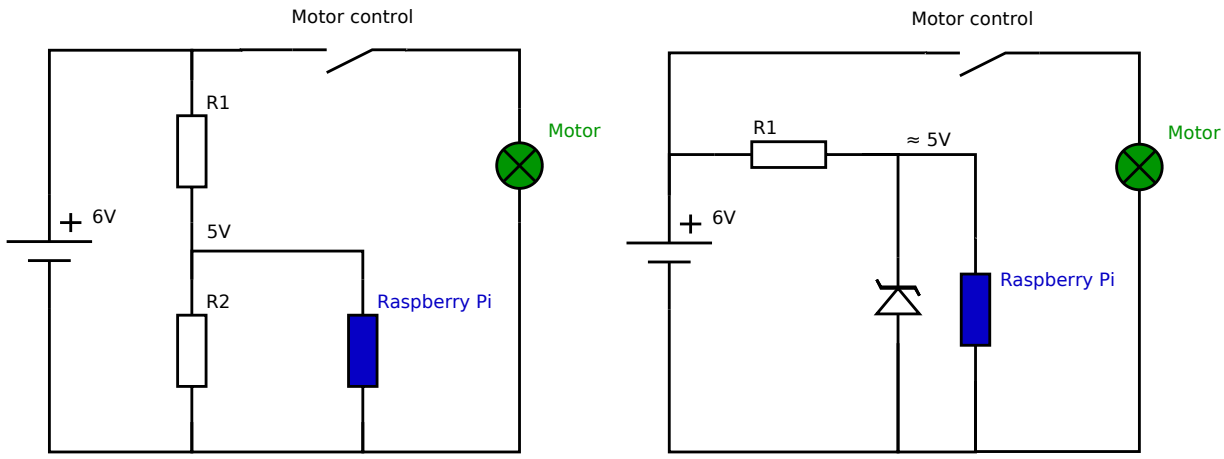
Voltage Divider

A voltage divider is a very common method to decrease the voltage. A basic voltage divider is a circuit of two resistors (R1 and R2) and the load (see figure 3.7(a)). The resistance of each resistor defines the resulting voltage on the load $V_L = \frac{R2}{R1+R2} * V_{CC}$.

A voltage divider does not work in the context of this thesis because of various reasons. First of all, the resistors limit the current that can flow into the load. Since the Raspberry Pi needs a high current, those resistors must have a low resistance. This would result in a high power consumption through the resistors.

Another problem is the power source. The actual voltage differs between full and empty batteries since the power is delivered by batteries. Unfortunately, the voltage would be divided in either case. So it would be too high on full batteries or too low on empty batteries.

A third problem is that the motors are connected to the batteries as well. Therefore they would pull down the available voltage while running and, thereby, cut the power supply for the Raspberry Pi. All together this would be a very wasteful and unreliable power supply.



(a) Simple voltage divider circuit. (b) Simple Zener diode circuit.

Figure 3.7.: Example circuits with a voltage divider or Zener diode.

Zener Diode

The Zener diode is a special diode that restricts the current flow in one direction but allows the flow in the opposite direction if a certain breakdown voltage is exceeded. The problem with this component is that it is too inaccurate. A common Zener diode has a breakdown voltage of 5.1V with a precision of about 10%. In the worst case this Zener diode would permit a voltage of about 5.6V, which is too high for the Raspberry Pi and would damage or even break it.

7805 with additional batteries

With two additional batteries there is a total of 6 batteries and, thereby, 9V available for the 7805 voltage regulator. In this case, the 7805 works well but the additional batteries cause new problems. The car is not build to carry so much equipment and weight. Therefore, the additional battery case had to be mounted somewhere on the car. Another problem is the weight of the batteries because the motors, especially the front motor, are already fully stretched with the additional weight by the Raspberry Pi and interlayer. Even more weight would result in even weaker motor responses or no responses at all.

Low Drop-out Voltage Regulator MIC29300-5

The low drop-out voltage regulator MIC29300-5 has shown to be the best solution for this task. It can provide a stable 5V signal if it has an input of at least 5.25V with a current of 1.5A which is enough for the Raspberry Pi and its appliances. If the current is lower, the input voltage can even be lower as well, for example, if the wireless adapter is removed or a model A Raspberry Pi is used.

The MIC29300-5 is easy to use because it has the TO-220 form factor, hence it has only three pins. The left one is the input, the middle one is for GND and the third is the output.

The Raspberry Pi has two voltage test points to check the actual on-board voltage while the Rasperry Pi is running. This is really helpful if the power supply is not reliable yet and the Raspberry Pi does not work properly. Those test points are labelled *TP1* and *TP2* as shown in figure A.4.

3.3.3 Final Wiring

The schema of the entire interlayer is shown in figure 3.8. The interlayer is located in the middle of the image and on the right is the Raspberry Pi. Left and right of the interlayer are the motors of the car. On the bottom of the schema is the battery case of the car that holds four AA batteries. This is the main power source for the entire circuit. Directly behind the power source is a switch that controls the entire circuit and thereby the power supply for the Raspberry Pi and the motors. Behind the switch is a capacitor that is supposed to stabilize the power supply for all components.

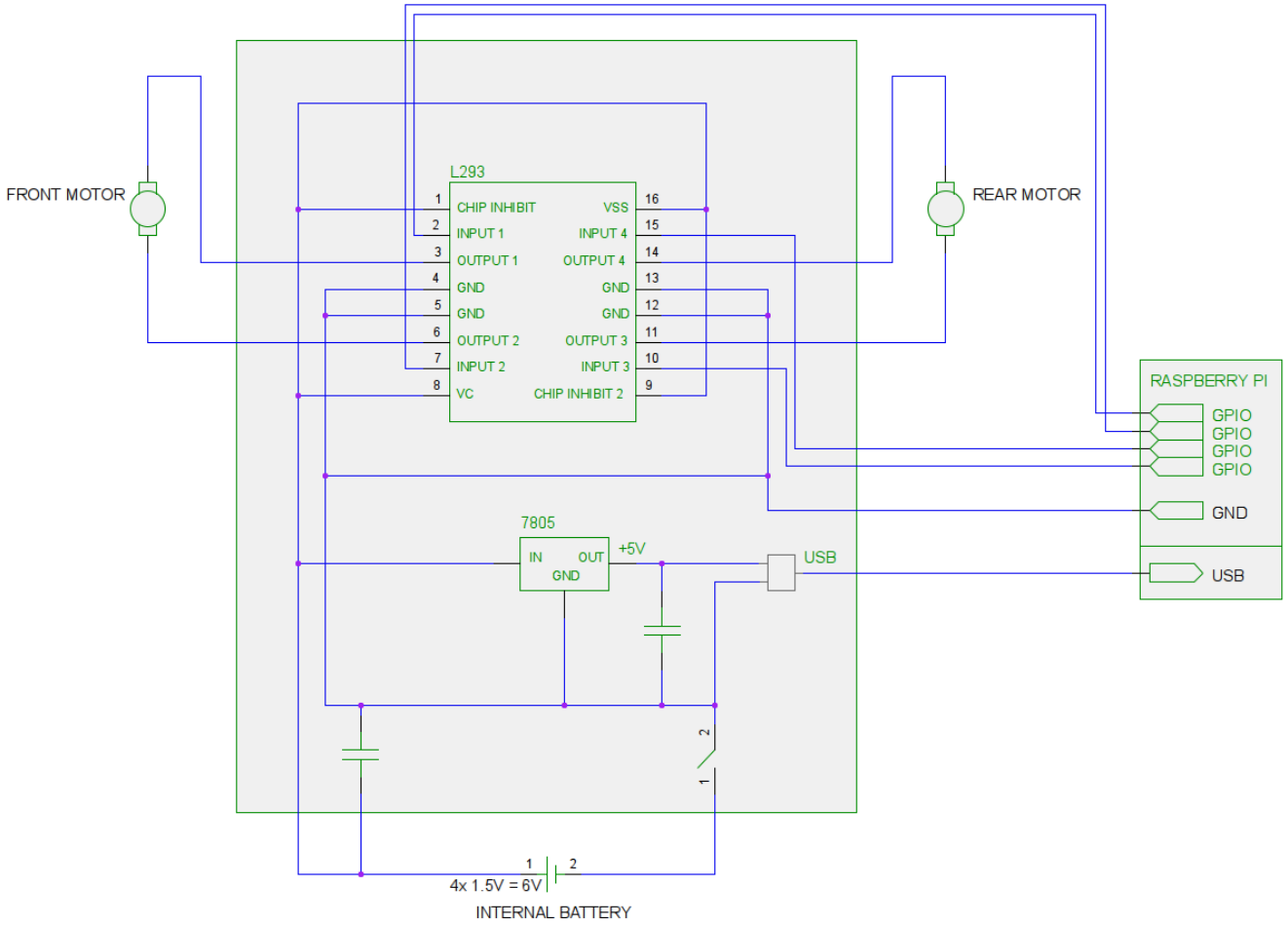


Figure 3.8.: Schema of the interlayer.

At the lower end of the interlayer is the voltage regulator for the Raspberry Pi. It has three pins which are connected to V_{CC} and GND. The third one is the output pin that is bound to a capacitor and to a USB cable that powers the Raspberry Pi. The capacitor is not crucial but makes the power supply more stable. The Raspberry Pi can also be powered by an external power source to save battery energy. In this case, the USB cable of the interlayer is unused.

Enable for the front motor	**1,2EN**	**V_{CC1}**	power input to minimize electrical potential dissipation
First direction for the front motor	**1A**	**4A**	Second direction for the rear motor
First power source for the front motor	**1Y**	**4Y**	Second power source for the rear motor
Common ground	**GND** **GND**	**GND** **GND**	Common ground
Second power source for the front motor	**2Y**	**3Y**	First power source for the rear motor
Second direction for the front motor	**2A**	**3A**	First direction for the rear motor
Power input (bound to the batteries)	**V_{CC2}**	**3,4EN**	Enable for the rear motor

Table 3.4.: Integrated circuit L293D wiring (motor control).

At the upper end is the motor controlling chip (L293D). It has 16 ports as shown in figure 3.6, and table 3.4 shows what these ports are supposed to do. This chip can handle either four motors in one direction or two motors in both directions. Because we need to control two motors in both directions, the motor enable pins (`1,2EN` and `3,4EN`) are not required and therefore bound to V_{CC} (+6V). If a motor is running or not, is specified by its two direction pins (`1A` and `2A` or `3A` and `4A`), see table 3.2. If a direction pin is set to GND, then its corresponding output is GND as well, for example if `1A` is set to GND then `1Y` outputs GND. If `1A` would be at +3.3V, `1Y` connects though to V_{CC2}, which is directly bound to the batteries. When both direction pins for one motor have the same value, the motor is either connected to GND twice or to V_{CC2}. In both cases the motor is turned off.

The first and last two ports are directly bound to the battery. The V_{CC2} port provides the power that is used to drive the motors. V_{CC1} is normally used to minimize the electrical potential dissipation but this is only required for the enable pins (`1,2EN` and `3,4EN`). Since they are not used in this context, V_{CC1} and the enable pins are connected to the batteries just like the main power source V_{CC2}. Finally the `Y` outputs are connected to the motors and the `A` input to the Raspberry Pi.

4 Object Detection

Object detection is required to determine the car's position relative to the target object. Images taken by the webcam are the only input that is available in the context of this thesis. Therefore those images have to be processed to extract information. That is what object detection is used for. It is supposed to find the target within the scene. Based on this information it is possible to estimate the position of the car relative to the target.

The program is written in C++, because it is the standard for embedded and low level programming. Additionally, C/C++ has many other advantages, for instance:

- It is fast, efficient and mature.
- It is compiled and, therefore, does not need a virtual machine as for instance Java would.
- It is a high-level programming language with low-level support.
- It is statically typed and provides object orientation.
- It is very popular and there are many libraries available.

4.1 Build Environment

The software consists of multiple source files and has some dependencies like the OpenCV library which is used for image processing. To make building the software as simple as possible, a program like CMake[1] is required. It manages the build process by creating the necessary Makefiles for the standard C/C++ compiler[2]. Roughly speaking, those Makefiles describe what the compiler should do and what it will need to do that.

Another advantage of CMake is its platform independence. This means it makes it is possible to compile the program on a desktop PC to check for any syntax errors before starting the compilation process on the Raspberry Pi. This helps a lot since the compilation takes some time, especially on the Raspberry Pi.

Listing 4.1: CMake file (extract).

```
1 find_package( OpenCV REQUIRED )
2 add_executable( ReinforcementPi
3         src/main.cpp
4         [...]
5 )
6 target_link_libraries( ReinforcementPi ${OpenCV_LIBS} )
```

Listing 4.1 shows an extract of the `CMakeLists.txt` file. Lines 1 and 6 tell CMake that the OpenCV library is required for this build and the resulting `ReinforcementPi` binary has to be linked against it.

[1] `www.cmake.org`

[2] A compiler is a program that translates the human readable source code into executable code.

Line 2 tells CMake what the name of the resulting binary should be. The source files, which should actually be compiled, are added in line 3 and below.

4.2 OpenCV

OpenCV[3] (Open Source Computer Vision) is an open source library mainly developed by Intel[4] and Willow Garage[5]. It is licensed under the BSD license and available for C/C++ programming language, but also for some others like Python.

The following command will install the library including their development files:

```
1 $ sudo apt-get install libcv-dev libopencv-dev
```

This needs about 170MB of space and will take a while to install because it has many dependencies.

4.2.1 Getting a Picture

Now, gathering a picture is pretty easy, at least in theory. There are two different ways to receive an image as shown in listing 4.2.

Listing 4.2: Ways of capturing an image with OpenCV.

```
1  cv::VideoCapture *capture = new cv::VideoCapture(0); // 0 -> default webcam
2  cv::Mat frame;
3
4  /** Possibility 1 **/
5  bool readSuccess = capture->read(frame);
6  // or
7  *capture >> frame;
8
9  /** Possibility 2 **/
10 bool grabSuccess = capture->grab();
11 bool retrieveSuccess = capture->retrieve(frame);
```

First of all, the webcam is opened by creating a new instance of the `cv::VideoCapture` class with `0` as the only parameter. By the time the camera gets opened, the first error type `VIDIOC_QUERYMENU` is displayed as shown in listing 4.3. Those errors are not critical since the program execution continues without further errors of this type. In line 2, a new `cv::Mat` is being created. This is the place where the new image will be stored. `cv::Mat` is a matrix datatype that is provided by OpenCV. Lines 5 and 7 show a possibility to get an image. Both lines are equivalent, with the small difference that the first one returns a bool value, which should be *false* if no image was grabbed. Those functions do internally call the `grab` and `retrieve` functions as shown in possibility 2 in lines 10 and 11. If they are called individually, each function returns a bool value that should indicate whether any problems occurred or not. Unfortunately,

[3] www.opencv.org
[4] www.intel.com
[5] www.willowgarage.com

this does not work. `grabSuccess`, `retrieveSuccess` and `readSuccess` are always *true*, regardless of an image was grabbed or not. This behaviour is probably caused by the problem that also triggers the `VIDIOC_QUERYMENU` errors. Each time, no image could be grabbed, a "`select timeout`" error occurs as shown in listing 4.3.

Listing 4.3: Errors while initializing the camera and grabbing an image.

```
1 $ ./ReinforcementPi
2 VIDIOC_QUERYMENU: Invalid argument
3 [... repeated 20 times ...]
4 VIDIOC_QUERYMENU: Invalid argument
5 select timeout
6 select timeout
7 [...]
```

It is easy to handle the "`select timeout`" errors by checking each image for its range of minimum and maximum brightness values. If they are equal, an invalid image was returned.

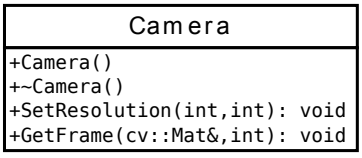
Unfortunately there is another odd behaviour. The first image that is not just black contains nothing but noise. Comparing the minimum and maximum brightness does not work either, because its darkest value is pure black and its brightest is pure white. This image is probably delivered from an internal uninitialized buffer. To avoid this problem, this image has to be dropped while initializing the camera.

The buffer causes another problem. It seems to hold four images. In case a new image shall be grabbed, the oldest image within the buffer is actually returned and the free slot within the buffer is filled with a new image based on the FIFO (First In, First Out) strategy. The actual returned image is potentially no longer up to date since it was taken in the past and then stored in the buffer.

In the context of this thesis this behaviour is very problematically. The learning algorithm requires current images of the environment because all decisions are made based the the current environments state. The upcoming actions that will be chosen are suboptimal if the processed image does not show the actual state. To handle this issue, each time an image shall be taken, actually five images are grabbed but the first four images are dropped immediately and only the last image is going to be forwarded.

Implementation

There is a special class, called Camera that gathers all those workarounds and abstracts from those problems to make further coding easier.



As visualized, the Camera class has a constructor and destructor (`Camera()` and `~Camera()`). The constructor opens and prepares the camera. Thereafter, it adjusts the resolution of the camera and rejects the first image since this is the noisy one. The destructor releases the camera again to make sure

it is not locked although the program itself terminated already. The function `SetResolution` is used to change the resolution and, thereby, the resolution of the future images. The parameters are used to pass in the desired width and height.

The most important function is the last one. It is used to read a current image from the camera and store it in a given reference to an OpenCV matrix object (`cv::Mat`). This function drops four images and returns the fifth to make sure the image is up to date and it also checks the brightness values. All images returned by this function are valid and current images. The second parameter is used to specify the desired color space. In the context of this thesis, this function is only used with the parameter `CV_RGB2GRAY` so the image will be transformed into gray scale.

4.2.2 SURF

The object detection algorithm that is used in this thesis is called SURF (Speeded Up Robust Features). SURF was published by Herbert Bay et al. in 2006 and is an improvement of the widely used SIFT (Scale-invariant feature transform) method, which was published in 1999. The authors of SURF claim it to be faster and more robust than the existing methods. "SURF approximates or even outperforms previously proposed schemes with respect to repeatability, distinctiveness, and robustness, yet can be computed and compared much faster."[6] Another paper by Luo Juan and Oubong Gwun compared SIFT and SURF with a similar result:

> "SIFT is slow and not good at illumination changes, while it is invariant to rotation, scale changes and affine transformations. SURF is fast and has good performance as the same as SIFT, but it is not stable to rotation and illumination changes." Juan and Gwun (2010)

Since the resources of the Raspberry Pi are limited and object detection is a very frequent task, SURF is the better choice in contrast to SIFT. The rotation *weakness* compared to SIFT does not matter so much, because the object within the scene is always at the same angle and does not rotate.

SURF is patented in the US[7] but can be freely used for educational, research and non-commercial purposes if this copyright notice is passed on. OpenCV, which is licensed under the BSD license, includes a rewritten SURF implementation that is used in context of this thesis. Anyway, the entire SURF license is attached to this thesis to make sure no licenses are disregarded (see appendix A.5).

SURF can detect the position of a predefined object within a scene. The object and scene are given as standard monochrome images. As mentioned, rotation, scale and affine transformations do not affect the detection result, at least in theory. To achieve that, SURF is divided into three main steps which are executed on the object image as well as on the scene image:

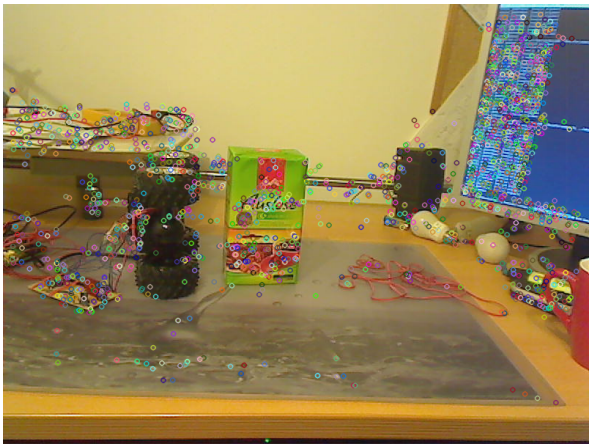
1. **Detection of important points so called *interest points***
   "The search for discrete image point correspondences can be divided into three main steps. First, 'interest points' are selected at distinctive locations in the image, such as corners, blobs, and T-junctions. The most valuable property of an interest point detector is its repeatability. The repeatability expresses the reliability of a detector for finding the same physical interest points under different viewing conditions." Bay et al. (2008)

[6] See Bay et al. (2008).
[7] Patent number: US2009238460

2. **Description of the detected interest points using their neighbourhood**
   "Next, the neighbourhood of every interest point is represented by a feature vector. This descriptor has to be distinctive and at the same time robust to noise, detection displacements and geometric and photometric deformations." Bay et al. (2008)

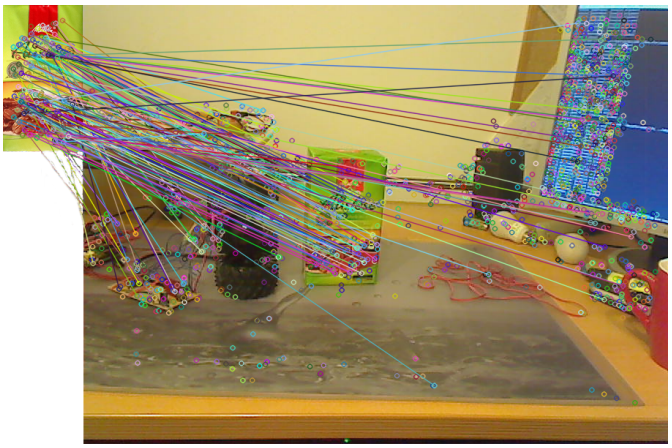
3. **Matching of the descriptors between object and scene images**

(a) Object image.



(b) Scene image.

(c) Enlargement of the object within the scene image.



(d) Matches between object and scene image.

(e) Detected object corners within the scene.

Figure 4.1.: SURF object detection example.

Figure 4.1 is a visualisation of those steps. Image 4.1(a) shows the real object image. This is the exact image that is also used by Raspberry Pi to detect the target. All images here are colored for demonstration purposes although the SURF algorithm operates only on monochrome images. Images 4.1(a) and 4.1(b) show the interest points that were found by step 1 of the SURF algorithm. Image 4.1(c) is an enlargement of the object in image 4.1(b). The shape of the interest points in those images is similar. Smooth surfaces like the background in the scene image do not have interest points because there are no corners or edges.

Step 2 is not visualised here because it is just an internal step.

Image 4.1(d) demonstrates step 3. All matching interest points between object image and scene image are connected by a line. Remarkably many lines link between object image and the actual object in the scene.

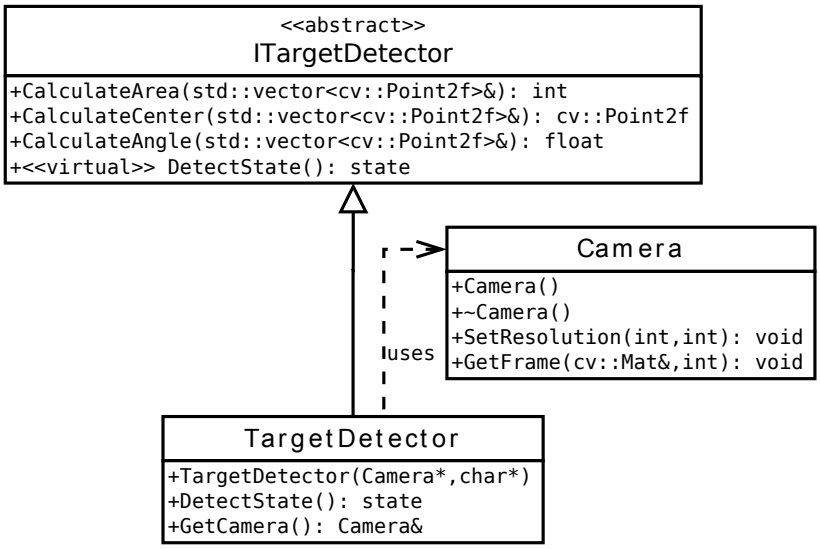
Finally the algorithm returns four points, that describe the position of the located object as visualized in image 4.1(e).

Based on the detected corners of the object, it is possible to calculate the geometrical area, the center point of the object and the angle at which the observer looks at the object (see A.6 for examples). The area is a measure for the distance to the object. The closer the car gets, the bigger is the area of the object within the scene. Therefore, the learning algorithm needs to maximize the area. The second inducible value is the position of the object within the scene image. This calculation is done based on the center point of the detected object. With this center point it is possible to determine the horizontal deviation within the scene.

Since the target object is fixed in context of this thesis, the image of this object does not change as well. Therefore steps 1 and 2 are only executed once for the object image to save computational effort. For each new scene image, all three steps are executed whereby the precomputed values of the object image will be used in step 3.

Implementation

There is a specialized class in the source code of this thesis to abstract from these three steps and automatically compute the values for the object image. This class is called `TargetDetector` since it is supposed to detect the target object.



`TargetDetector` is a class that inherits by the abstract class `ITargetDetector`. The super class `ITargetDetector` is abstract. That means, it is not possible to instantiate this class because it defines an abstract function, in this case `DetectState()`. This function is just declared, but not implemented. That is why any child class has to implement this function or needs to be abstract as well. Additionally, `ITargetDetector` offers some common functions to calculate the area, center point and angle of the detected object based on the corner points of the object within the scene.

The list of corner points looks like $C = (c_1, c_2, \ldots, c_i)$ where each point is a tuple of x and y values (see figure 4.2). Usually this list is of size 4 since there are exactly 4 object corners. Nevertheless, `CalculateArea()` and `CalculateCenter()` work with any number of corners if $|C| > 1$ is satisfied.

$$\texttt{CalculateArea}(C) = \frac{1}{2}\sum_{i=2}^{|C|} |x_{i-1}y_i - y_{i-1}x_i| \quad \text{where } c_i = (x_i, y_i)$$

$$\texttt{CalculateCenter}(C) = (\frac{1}{|C|}\sum_{(x,y)\in C} x, \frac{1}{|C|}\sum_{(x,y)\in C} y)$$

Figure 4.2.: Functionality of common functions `CalculateArea` and `CalculateCenter` of class `ITargetDetector`.

The `TargetDetector` class requires an instance of the `Camera` class and a char array or in other words a string. The `Camera` instance is used to gather new images each time `DetectState()` is called. The char array is supposed to be the path to the object image. This image is only required once because the constructor executes the SURF steps 1 and 2 on the given image and keeps the result for future use.

The function `DetectState()` is the main function of this class. It gathers a new scene image and executes all SURF steps with the use of the previously stored results of the object image. When the object is detected within the scene, four points are returned, which describe the border of the object (see image 4.1(e)). The new state can be calculated from that, by the help of the common methods provided by the `ITargetDetector` class (see figure 4.2).

The state space is defined by the set $\{(area, direction)\} \subseteq \mathbb{N} \times \mathbb{Z}$. States are implemented as a structure composed of two variables, *area* and *direction*, see listing 4.4. Furthermore, each state *knows* whether it is a final state (line 5), and can calculate the reward respecting its previous state (line 6). A state is a final state if its `area` value exceeds a predefined threshold.

Listing 4.4: Snippet of the `state` structure.

```
1 struct state {
2         int area;
3         int direction;
4
5         bool IsFinalState() const [...]
6         int Reward(const state last_state) const [...]
7         [...]
8 };
```

The mapping from the detected objects position and size to a state instance is done in the function `DetectState()` of class `TargetDetector`. The `area` value of the state is calculated using the

equation 4.1, whereby the function `CalculateArea()` is used to calculate the geometrical area of the detected object within the scene.

$$area = \left\lceil \sqrt{\frac{\texttt{CalculateArea()}}{1000}} \right\rceil \in \mathbb{N} \tag{4.1}$$

The square root is required to maintain equal distances between different area ranges. This is visualized in figure 4.3. Both plots show the geometrical area of the detected object based on the formula $A = width * height$, whereby the x-axis shows the width of the object and the y-axis shows the height. The contours separate the geometrical area space into 15 steps. Based on the non-linear growth of the geometrical area, the contours of the naive approach $width * height * 10^{-3}$ in the left plot are unequally. The right plot shows the same calculation but with an additional square root $\sqrt{width * height * 10^{-3}}$. With that, the contours are consistent.

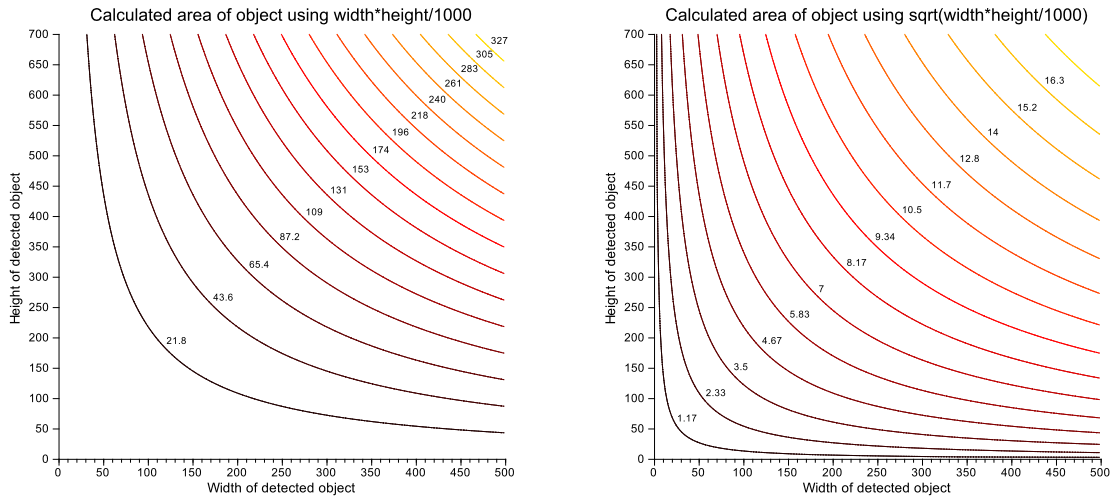


Figure 4.3.: Comparison between geometric area of detected object and calculated state area value.

The effect of equal area sizes is that the states magnitudes are comparable, regardless of whether the car is close to the target or far away. Without the square root, there would be more states close to the target and those would be smaller than states further away.

The *direction* is calculated by using just the x-coordinate of the center point of the detected object, as shown in equation 4.2. The y-coordinate does not matter, since the car cannot move within this dimension. $\frac{1}{2}CAM\_IMAGE\_WIDTH$ is the center of the scene in x-direction. If the center point of the detected object is exactly at this position, the resulting *direction* is 0. The deviation from the center of the scene is divided by 100 to reduce the number of possible states and to increase the tolerance for detection deviations.

$$(x, y) = \texttt{CalculateCenter}()$$
$$direction = \left\lfloor \frac{x - \frac{1}{2} CAM\_IMAGE\_WIDTH}{100} \right\rfloor \tag{4.2}$$

where *CAM_IMAGE_WIDTH* is the width in pixels of the scene image

Target object

The target object in this thesis is an ordinary coffee bag. The coffee bag has some advantages in this context, because it

- has a good size,
- is easy to set up since you just place it on the ground,
- will not fall over if the car drives against it,
- is well suited to be detected by the SURF algorithm.

The advantages for the SURF algorithm are that it is not too small and, within its cover, it has a lot edges and corners. In other words, it has many possible interest points. The color of the object does not matter since SURF only works on monochrome images.

Problems

Noisy images

Since images taken by the webcam are noisy[8], the position and size of the detected object varies. Unfortunately, the detected points differ even if neither the camera nor the object was moved (see figure A.6 for examples). Those differences increase with increasing distance to the object. In other words, the detection quality decreases with increasing distance.

Image size

The SURF algorithm works better with increasing image resolutions, but bigger images also require more memory and the computational effort increases rapidly. A resolution of 640x480 pixels has shown to be a good compromise, although a higher resolution would result in less object detection errors and a increased operational range of the car.

[8] There are some techniques to reduce this noise like blurring or bilateral filtering, see section A.2.2.

5 Reinforcement Learning

Reinforcement learning is an effort to copy how animal learn. Especially young animals learn by interacting with their environment and judge the result, so called *trial-and-error learning*. Reinforcement learning tries to imitate this behaviour to some extent.

> "Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards."
> Sutton and Barto (1998), p. 4

The scenario was that the car is placed somewhere in front of the target object and the Raspebrry Pi can perceive this object by its webcam. With this information it is supposed to learn how to control the motors correctly to drive towards the object.

As from now the car, Raspberry Pi and all its components are called *agent*. The agent's task is to interact with its *environment*. The environment is, in this case, limited to the surface the car stands on and the target object. Mathematically speaking, the environment has only two dimensions since the agent can just move forth and back, left and right but not up and down.

The interaction between agent and environment is restricted as shown in figure 5.1. The environments current condition is modelled as a *state* it is in. The agent chooses actions considering the current state. Not all actions must be available in any state, so there could be a case where there is a state that has only a subset of actions available. An action can potentially modify the environment and thereby result in a state change.

Executing an action produces a numerical reward that is a measure of the actions quality in connection with the state it was invoked in and the state the agent ended up after the action was executed.

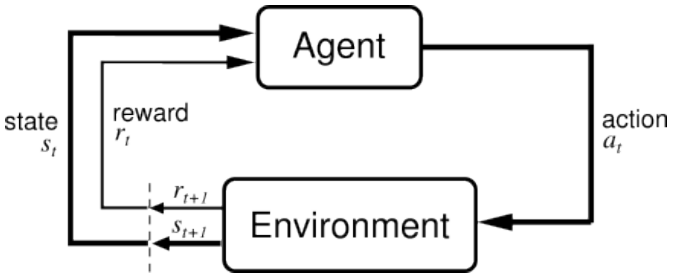


Figure 5.1.: Agent-environment interaction.

Source: R. S. Sutton and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press, p. 44

The current state is the only information that is available to the learning process about its environment, at any point in time. Therefore, it is important that the state contains all important information about the current condition of the environment.

5.1 Components

A reinforcement learning model is build of these main components:

- **States**
  Set S of states the agent and environment can be in.
- **Actions**
  Set A of actions the agent can invoke. This set can be restricted depending on the current state.
- **Rewards**
  A function that provides numerical rewards for state transitions. It is used to estimate the quality of action a in state s based on the state change it causes.
- **State Values**
  The value map memorize what outcomes an agent expects for given states.
- **Policy**
  A policy is a structure that maps states to actions. Roughly speaking, it defines what action to take in a specific state.
- **Model (optional)**
  A model of the environment and agent predicts the new state s' when action a is invoked in state s. The model can be probabilistic or unavailable.

5.1.1 States

The only variable element in this scenario is the agent itself, because the environment does not change, but the agent can move within the environment by invoking actions. The agent can perceive its position relative to the target that is detected using the object detection. The state that is perceived by the object detection is actually the state that is also used in the learning algorithm. There are other imaginable inputs as well, for instance the current rotation angle of the front axle which can, unfortunately, not be measured with the current hardware set up. The only input in this thesis is the image that was taken by the webcam and processed with the object detection module, see section 4.2.2 for more information.

The state is a tuple composed of two values and can be written like (*area*, *direction*):

- **area**
  A measure for distance to the target. If the car get closer to the target, the size (area) of the target object increases within the scene taken by the webcam. If it departs, the size of the object decreases as well. Therefore, a larger area value means the agent is closer to the target.
- **direction**
  A measure for the viewing direction (left-right-deviation) of the car relative to the target object.

Theoretically, there is also a third value describing the viewing angle. This value can be calculated but due to noise in images taken by the webcam this value is too inaccurate to be useful.

The state space in this case is finite due to the capabilities of the webcam. The target cannot be detected if the agent is too far away. Therefore, the area the agent can operate in, is limited backwards. In the front, the operational area is limited because the agent can only reach its target or miss it. Missing would result in the special state (0,0) that is used if the target could not be detected any more. Furthermore, the operational area is also limited to the sides because the webcam does not perceive the target object, if the car drove to far either left or right or it rotated on the spot. In this case, the same special state is used as mentioned above.

State $(0,0)$ is used as the special state since it represents the situation where there is no information about the position of the agent relative to the target because the agent cannot perceive the target any more. Getting into this state should be avoided by the agent.

The set of states can be accurately described by:

$$S \subset \mathbb{N} \times \mathbb{Z}$$

$$S = \{(a,d) : a \in [1,16] \text{ and } d \in [-4,4]\} \cup \{(0,0)\},$$

where a is the area of or distance to the target and d the direction.

5.1.2 Actions

Due to the hardware constraints, there are four elemental actions like forward, backward, rotate left and right and four aggregated actions like go forward or backward and steer left or right. For a detailed explanation see section 3.3.1. Finally, there is a last action that just does nothing.

The set of actions is in the context of this thesis defined by $A = \{\mathtt{f}, \mathtt{b}, \mathtt{l}, \mathtt{r}, \mathtt{L}, \mathtt{R}, \mathtt{K}, \mathtt{E}, \mathtt{0}\}$. This set could be restricted depending on the current state. This is not done in this context since the agent cannot harm itself or anything within its environment. Besides that, it is more interesting to see that the agent learns what actions are useless in which states instead of manually specifying that.

5.1.3 Rewards

A reward is a numerical value that compares two states with another. This is crucial for learning because this is the feedback the agent gets after executing an action. After the execution of an action, the environment can potentially be in a different state s', but that is not compelling. A reward is also gained if $s = s'$. The reward is calculated based on the old and new state, not on the action.

Most of the time, the reward an agent receives is negative since only final states do have a positive reward. This is called *delayed rewards* because previous states give a negative reward although they are maybe a good step on the way to a final state. That last transition into the final state will receive a positive reward. Therefore, that positive reward has to be back-propagated to the previous states.

The scenario that is handled by this thesis encloses delayed rewards as well, because all non-final states are technically the same and will have a negative reward. The final state, just in front of the target, will have a positive reward that needs to be back-propagated to previous states.

5.2 Markov Decision Process

A reinforcement learning problem is a Markov decision process (MDP) if the *Markov property* holds. The Markov property requires, that the transition probability of action a to lead to state s', when invoked in state s, depends only on the state s, and not on any previous state. In other words, it does not matter how the agent got into state s to specify the probability that an action a leads to state s' when invoked in state s.

To achieve that, it is important that a state instance contains all modifiable elements of the agent and environment. If not so, previous states could have changed one of those unrepresented elements, which results in a different behaviour in a future state. Thus, this elements are not negligible and hurt the Markov property.

> "Note that although both transitions and rewards may be probabilistic, they depend only upon the current state and the current action: there is no further dependence on previous states, actions, or rewards. This is the *Markov property*. This property is crucial: it means that the current state of the system is all the information that is needed to decide what action to take–**knowledge of the current state makes it unnecessary to know about the system's past**." Watkins (1989), p. 39

All in all, the Markov property demands that the entire configuration of the agent and environment is fully represented by any state.

5.3 Q-Learning

The learning algorithm that is used in the context of this thesis is called *Q-Learning*, which is a model-free *Temporal-Difference (TD)* algorithm. TD learning methods combine the ideas behind *Monte Carlo* and *dynamic programming* methods. Therefore, a Q-Learning algorithm does not need a model just like Monte Carlo methods. Furthermore, TD methods update the state value directly after each step like dynamic programming methods.

Q-Learning does only work with MDP's, because all values are calculated based on the current state. Therefore, each state instance must represent the entire configuration of the agent and environment.

The basic Q-Learning update is defined by:

$$Q'(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{old value}} + \alpha \left[\underbrace{r(s,s') + \gamma \max_a Q(s',a)}_{\text{learned value}} - \underbrace{Q(s,a)}_{\text{old value}} \right] \tag{5.1}$$

As shown in equation 5.1, a Q-Learning update consists of the old Q-value and a learned value. The learned value is the sum of the reward $r(s,s')$ the latest action received, and the expected future value $\max_a Q(s',a)$ based on the new state s'. Since the Q-value is supposed to converge to the optimal value, the old value needs to be subtracted from the learned value. There are two variables in this equation to fine-tune Q-Learning updates:

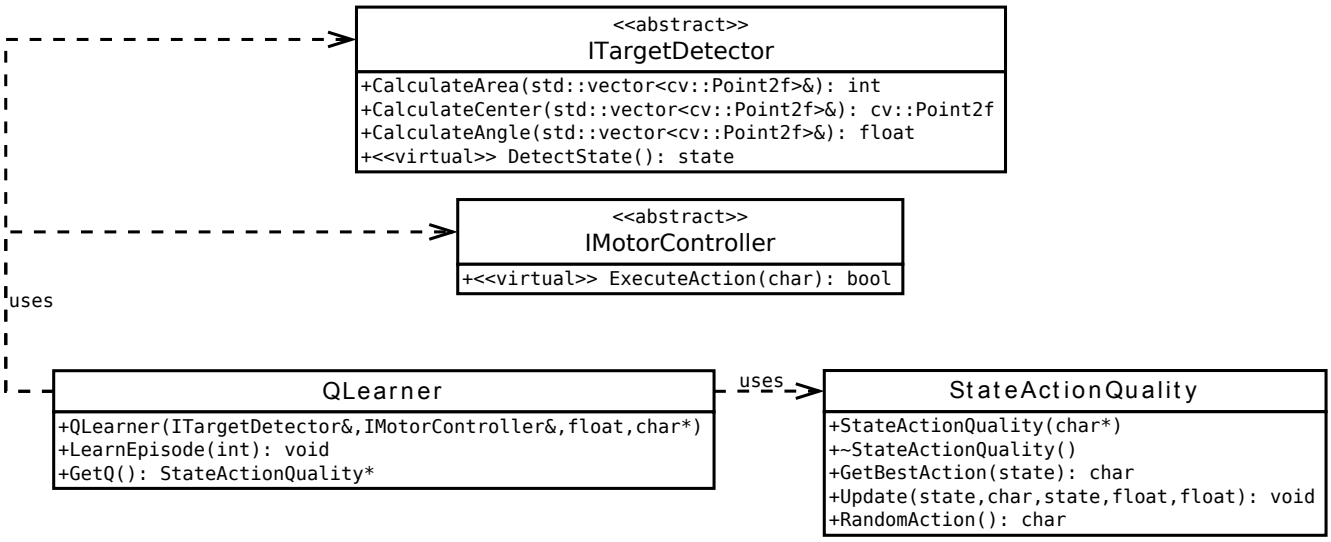
- **Learning rate α**
  The learning rate specifies how strong an update alters the existing Q value. With a learning rate value close to 1, the update value has a big effect on the new Q value, whereas a learning rate value close to 0 would affect the new Q value only slightly. The learning rate can be reduced as the agent becomes more experienced, to preserve older experience from being overwritten by random variations.

- **Discount factor γ**
  The discount factor defines how long-sighted the agent should be. With a discount factor close to 1, the agent would learn long-sighted, because the expected future reward is fully taken into account. A lower discount factor close to 0 would result in a short-sighted behaviour because the reward for the latest action is more important than the expected future reward.

Q-Learning as shown in equation 5.1 operates as an off-policy reinforcement learning method, because the *maximum* expected future value is used. In contrast, on-policy algorithms would use the expected future value based on the action that is chosen by the policy.

5.4 Implementation

The implementation of the Q-Learning algorithm is split into two classes: `QLearner` and `StateActionQuality`. The class `QLearner` is responsible for the learning sequence, whereas the class `StateActionQuality` manages the state values, chooses what action to take and updates the Q-value as shown in equation 5.1.



`QLearner` is the main class. To instantiate this class, a specialization of `ITargetDetector`[1] and `IMotorController`[2] is required as well as a `float` value and a `char` pointer. The `char` pointer is supposed to be the path to the *memory file*. This is the file where the learned experience will be saved. If this file exists, the prior experience is automatically loaded.

[1] See class `TargetDetector` in section 4.2.2.
[2] See class `MotorController` in section 3.3.1.

The `float` value is the *Explore-Exploit* ratio. This value defines the ratio between exploration of new knowledge and exploitation of existing knowledge. A value close to 0 forces a high exploration probability, whereas a value close to 1 forces the exploitation of knowledge. This parameter is comparable to the ϵ-greedy action selection strategy, whereby the *Explore-Exploit* ratio corresponds to ϵ.

In contrast to an exploration step, an exploitation step chooses the best known action for the current state if possible. This is not possible if there is no knowledge for the current state. In that case, an exploitation step chooses a random action like an exploration step always does.

Regardless of which strategy was chosen, after the execution of the selected action, the new state is detected and, based on that information, the Q-value can be updated. This is done by the `StateActionQuality` class. This class receives the `char` pointer containing the path to the memory file. The main part of this class, is its double mapping to store the knowledge:

```
1 std::map< state, std::map< char, double > > experience;
```

This double `std::map` can be thought of as the Q-matrix as used in equation 5.1. It has an independent field for each state-action pair. The initial value of each field is 0.

The `Update()` function does exactly what is described in equation 5.1. Hence, it requires the same values as used in the equation above: two states s, s' and an action a whereby the state s' is the current state that was observed after executing action a in state s. The last two parameters are the *learning rate* and *discount factor*. In the following source code snippet, the state s is called `last_state` and s' is called `new_state`:

```
1 double learned_value = new_state.Reward(last_state) +
2                 discount_factor * maxValue(new_state);
3 experience[last_state][a] = experience[last_state][a] +
4                 learn_rate * (learned_value - experience[last_state][a]);
```

The internal function `maxValue()` searches the maximal value that is known for the given state. The states reward is calculated considering the previous state by using a equation as shown in figure 5.2.

$$r(s,s') = r((a,d),(a',d')) = \begin{cases} 30-(d')^2, & a' >= 14 & (s' \text{ is a final state}) \\ -10, & a' = 0 & s' \text{ has area value of 0} \\ -1, & a < a' \vee |d| > |d'| & s' \text{ is an improvement against } s \\ -5, & else & \end{cases}$$

Figure 5.2.: Reward calculation equation.

The function `GetBestAction()` chooses the best action based on the given state. Internally, this is just a Q-matrix filter. This function is used, when the `QLearner` class chooses to do an *exploit*-step. In contrast, if an *explore*-step should be done, the function `GetRandomAction()` is used. It is supposed to choose a random action, whereby each action has the same likelihood. It is also imaginable to use

different likelihoods to make good actions be chosen more often compared to bad actions. This was used in early implementations but has been removed to make sure the car is actually learning without a given preference.

`LearnEpisode()` is the main function of class `QLearner`. It composes the object detection, motor control and class `StateActionQuality` as visualized in figure 5.3.

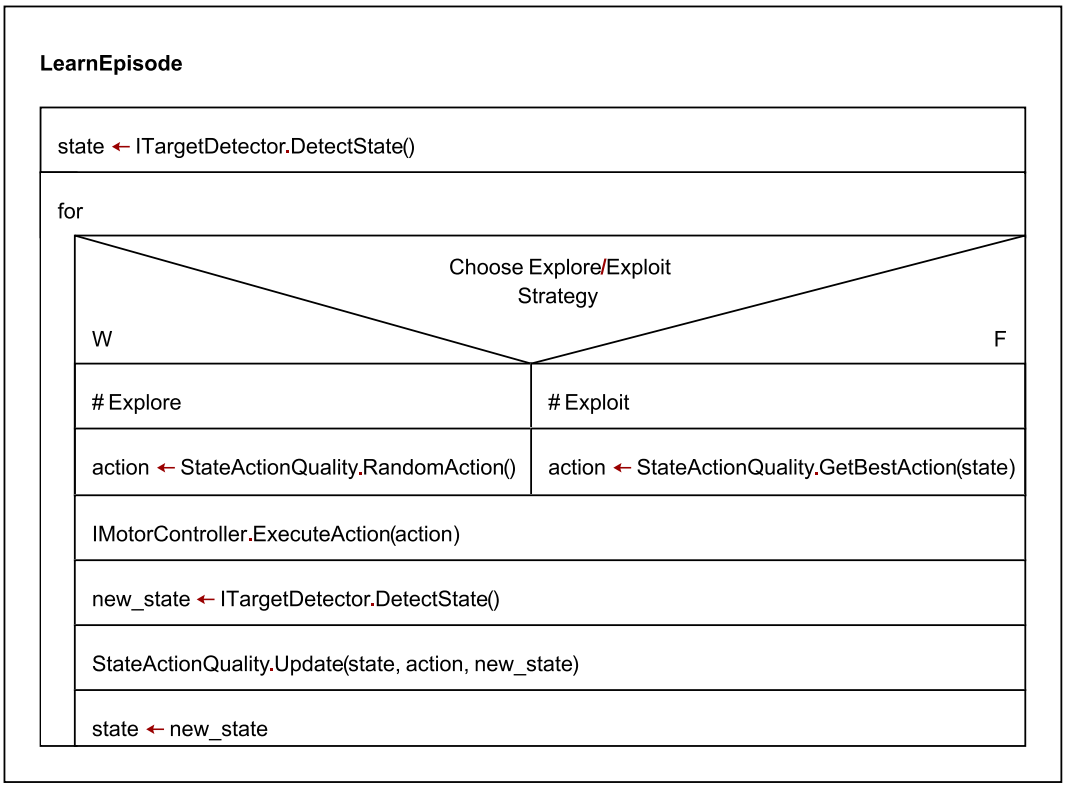


Figure 5.3.: Nassi–Shneiderman diagram of function `LearnEpisode` in class `QLearner`.

The first step is to detect the initial state. Afterwards, a loop is started that repeats for each learning step. The number of iterations of this loop can be limited using the *int* parameter of the function `LearnEpisode()`. For each learn step, the function chooses between an explore or exploit step. The *Explore-Exploit* ratio is used to influence this decision. After an action was chosen, it is executed by the `IMotorController` class. Thereafter, the agent and environment can potentially be in a different state. The new state is detected by using the `ITargetDetector`. Eventually, all information required to update the Q-value is available, because the start state, the action that was taken and the new state is known. By repeating this loop, the agent will improve its Q-value and can make use of the knowledge by choosing exploit steps.

Figure 5.4 is a movie demonstrating the final result. The Raspberry Pi is connected to an external power source because the batteries cannot supply the Raspberry Pi and both motors at the same time. Unfortunately, it is crucial that both motors can operate at once, because complex actions like `L` or `R` depend on that. In this case, the voltage drops below the acceptable threshold for the low drop-out voltage regulator and thus the power supply for the Raspberry Pi collapses. The batteries can only supply the Raspberry Pi and one motor at once.

In the first part of the movie, the car is learning. It starts without any prior knowledge. After about 20 minutes of learning, the car has enough experience to reach the target optimal. Therefore, an explore-exploit ratio if 1 was used to make sure no explore steps are used. Instead, each step executes the action that was learned to be best in the current state. After reaching the target, the car enters a state it was not in before and thus has no experience. In this case, the car does an explore step. This is marked in the movie.

Figure 5.4.: Demonstration movie of the car while exploring and exploiting.

This movie is also on Youtube: `www.youtube.com/watch?v=5ZjxUtCXSDs`

Problems

Steering

The biggest problem is the steering behaviour. Since the front motor has to lift the entire car, is requires a much power to do that. With full batteries, this works, of course, better than with batteries that are already used for some time. Unfortunately, batteries are already too low after being about half an hour in use. Therefore, action can have different outcomes, for example action 1[3], even within a very short timespan.

Another odd behaviour is that steering works better, if the motors did run just a few seconds ago. So if the agent starts operating, rotating does not work, but after some trails, it starts to work.

[3] Just rotate to the left.

Different surfaces

Another problem are different surfaces. The front axle rotation takes more time on a carpet than on a smooth surface like a desk. For instance, if the rotation works as intended on a carpet, it would result in a 450° rotation on a smooth surface. This means, the front axle is directly in front of the webcam and covers the target object.

Target out of sight

There is only one state to indicate that the target is out of sight. This is state is active in many different situations, for instance if the car is too far away or too close but also if the front axle blocks the view to the target object. Because those situations are very different, no good action can be found for this state. For example, when the car is too close, drive backward is a good idea but drive forward not. In contrast, when the car is too far away, drive backward is bad and drive forward good. In other words, the actions of *too close* are the complete inverse of the actions of *too far away*.

Since there is no other input expect the webcam image, especially no feedback by the motors, the car cannot do any predications if the target could not be detected.

5.5 Simulation

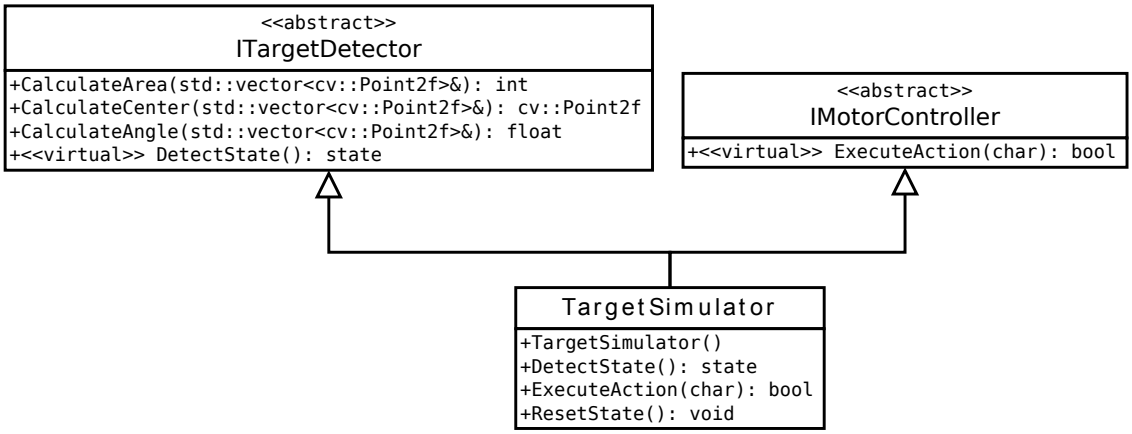
Since the agent has to try any action in each state multiple times, this can be extremely time-consuming. In the context of this thesis, there are 16 possible `area`-values and 5 `direction`-values, thereby $16 * 5 = 80$ possible states times 9 possible action. In total, the agent has to execute at least $80 * 9 = 720$ actions to have a basic knowledge of any action in any state.

Simulation can help to solve this problem because it is done just virtually, without real world interaction. Simulating the agent and environment has multiple applications:

- Verify the learning algorithm.
- Debug the program.
- Generate knowledge, which can, for example, be used and approved by the real agent and environment.
- The simulation can be used as a framework to test algorithm modifications or even different algorithms.

This simulated knowledge can be much closer to the optimum than knowledge that was learned by the agent, because the simulation can do more episodes in the same time than the real agent. Of course, the simulated knowledge is only as good as the model, that is used to simulate the agent and environment. When the model is not close enough to the reality, the generated knowledge is not usable by the real agent. Therefore, the aim is to model the agent and environment as good as possible.

Furthermore, the simulation should be using the same code as used for the real agent to ensure that the simulation really acts like the real agent. That is the reason why class `QLearner` expects instances of the abstract classes `ITargetDetector` and `IMotorController`.



The class `TargetSimulator` inherits from both classes and, thereby, acts as both, target detector and motor controller. This is really helpful, because the `TargetSimulator` is supposed to simulate a state and modify this state whenever an action shall be executed. This can be achieved by having the current simulated state and `ExecuteAction()` function within one class.

The class `TargetSimulator` picks a random valid state when being created and stores this state internally. That state can be accessed, as usual, by the function `DetectState()`. Whenever `ExecuteAction()` is called, the internal state is modified based on the given action and on random values. Randomness is used because the cars behaviour is not predictable in reality as well. To describe that in the simulation, the outcome of any action is calculated based on random values:

```
1 int v1 = rand() % 2, v2 = rand() % 2;
2 switch (action) {
3         case 'f': s.area += (v1+v2); break;
4         case 'b': s.area -= (v1+v2); break;
5         [...]
6 }
```

In line 1, two variables are initialized with a random value, either 0 or 1. Those values are used differently, depending on the action that shall be executed. In case of action `f`, both values are added to the `area` value of the current state. There are three possible outcomes of this operation: The `area` value does not change because both variables are 0, the `area` value is increased by 1 because one of the variables is 1 and the other one is 0, or both variables are 1 and the `area` value is increased by 2. The meaning by that is, that the *forward* action mostly will bring the car closer to the target, but sometimes there are detection errors or other circumstances so that the state changes not or stronger than normal. The other actions operate similarly, always based on those random variables.

Classes `QLearner` and `StateActionQualtity` do not need any modification to work within a simulation. In this manner, the simulation uses exactly the same algorithm as the real agent does.

5.6 Q-Value Animation

By the help of the simulation environment, animations can be created that document the learning progress of the Q-value. Such animations could also be created using the real agent, but with a much higher effort.

In the example as shown in figure 5.5, each frame visualizes the result of a simulation of 25 explorations. The first animation shows the best value of the entire Q-matrix. The following animations visualize the values of single actions, like `0` or `L`.

The shown graph does not reflect the simulated values absolutely accurately, because the mesh is calculated by the help of interpolation. Nevertheless, the shape corresponds to the real shape.

As time goes by, the values improve visible and advance towards an optimum.

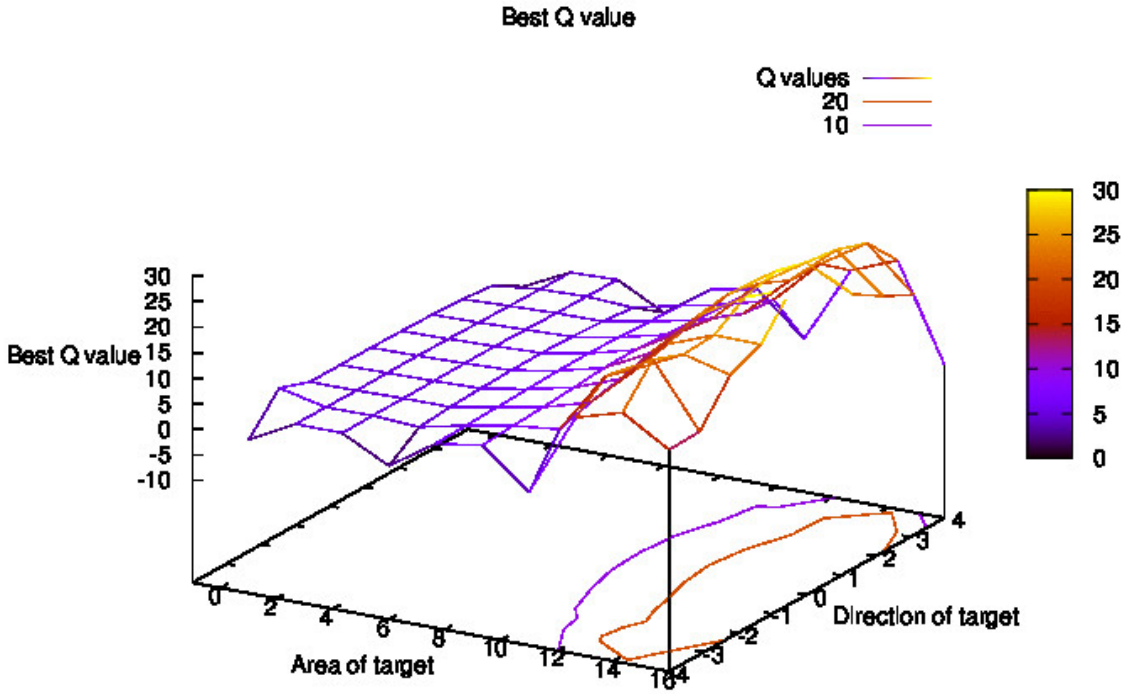


Figure 5.5.: Visualization of the Q-value while improving through simulated exploration.

This movie is also on Youtube: `www.youtube.com/watch?v=xEtVddFG7c8`

6 Lookout

During the work, some ideas came up, how the current set-up could be enhanced or upgraded.

Motor control

There is an open source library to control the GPIOs of the Raspberry Pi called *wiringPi*[1]. It has some advantages compared with the current implementation, for instance programs using *wiringPi* do not need to run as root user to access the GPIOs.

Front Motor

As mentioned, the front motor does not work reliable because it cannot perform a rotation at any time, with the same outcome. Sometimes, the motor is too weak to achieve a full rotation, after some trials this works suddenly. Additional, the time required to perform a full rotation differs. Unfortunately, there is no feedback by the motor. It could, for instance, indicate what the current angle of the front axle is.

This could be improved using a more sophisticated motor. First, it has to be stronger than the current motor. Second, and this is really important, it must give some feedback about the success of an instruction or indicate the current angle or similar information. This information would be really helpful for the learning algorithm to refine the current state of the car.

Power supply

Another idea is to use an external power source and rechargeable batteries to either power the entire car including the Raspberry Pi by the external power source or to charge the batteries while the car can operate as usual. After unplugging the power source, the car could run without any interruption on batteries again.

A addition to that would be a chip that measures the current charge of the batteries and provides that value to the Raspberry Pi. It could use this information to modify the motor control to make sure all action behave the same at all charge levels. Eventually, the Raspberry Pi could shut down if the batteries get to low to power the circuit properly.

Camera

A special webcam for the Raspberry Pi is available since May, 2013[2]. It is smaller and lighter than the webcam that is used in the context of this thesis but it has similar features.

In contrast to that, a *Microsoft Kinect* camera could improve target detection a lot, because it can actually sense depth and, thereby, measure the distance to the target. On the other hand is it probably too heavy for the car and consumes too much power for batteries. Another disadvantage is its price, because it costs about as much as the car, interlayer and Raspberry Pi together.

[1] projects.drogon.net/raspberry-pi/wiringpi
[2] http://www.raspberrypi.org/archives/3890

Raspberry Pi

Since the Raspberry Pi is not used to the full in the context of this thesis, it could be replaced by a model A Raspberry Pi that consumes roughly a third of the power that a model B consumes. Additionally, a model A Raspberry Pi is cheaper.

Learning

Right now, whenever the target object is out of sight, state (0,0) is active. This is the case when the car is too far away, too close or drove too far left or right. Actually, this state is active at any time the car can not see its target, therefore, also if the front axle blocks the view to the target object or the car is fallen over.

A possible solution for this is using additional markers in combination with a *Hidden Markov model*. For example, distinct markers could be placed all around the target so that the Raspberry Pi can perceive markers at any time, also if the target is out of sight. An additional marker on the ceiling would be helpful if the car was fallen over. Based on this set-up, a Hidden Markov model could be used to extract information from those markers to find the way back to the target. This could be done with the current low-cost webcam.

7 Conclusion

This thesis did evaluate the abilities of the Raspberry Pi as a mobile, low-cost, on-board computer. The aim was to do sophisticated image processing and learning on-board a rebuild RC car.

Thereby, three major tasks had to be solved. First, the hardware had to be prepared to be able to supply the Raspberry Pi and vice versa that the Raspberry Pi is in control of the car. This was done by using a low drop-out voltage regulator for the power supply and a particular integrated circuit component to control the car based on the low-level signals of the Raspberry Pi. Second, the Raspberry Pi needed to be able to perceive its environment. For that reason, a webcam was attached to the Raspberry Pi. By the help of this, the car could identify its current position relative to a target using an object detection algorithm. Third, actual learning was implemented based on the Q-Learning algorithm.

The result of this investigation is, that all computational work can be done on-board by the Raspberry Pi, without being stretched to its limits. Taken together, these findings do support a strong recommendation in using the Raspberry Pi even for expensive tasks. By doing so, many tasks could be solved by this low-cost microcomputer with no need for an off-board computer.

A Appendix

A.1 Hardware

	Model A	Model B (Revision 2)
CPU:	700MHz ARM11 (overclockable up to 1GHz)	
GPU:	Broadcom VideoCore IV (24 GFLOPS)	
RAM:	256MB	512MB
Video output:	Composite or HDMI	
Ports:	SD card slot 8x GPIO (among others)	
	1x USB 2.0	2x USB 2.0 10/100 Ethernet port
Power Supply:	5V ± 0.25V (USB specification)	
	300mA (1.5W)	700mA (3.5W)
Price:	US$25	US$35

Table A.1.: Raspberry Pi hardware specification and comparison.

A.1.1 Motor Control with Single Transistors

I also tried using a couple of MOSFETs. A MOSFET is a special kind of a transistor that is able to switch big potentials (for instance up to 36V). After some research I decided to try the IRFU 9024N and IRFZ 24N. These are very similar MOSFETs but the IRFU 9024N is a P-Channel and the IRFZ 24N is a N-Channel transistor. You see them in figure A.1.

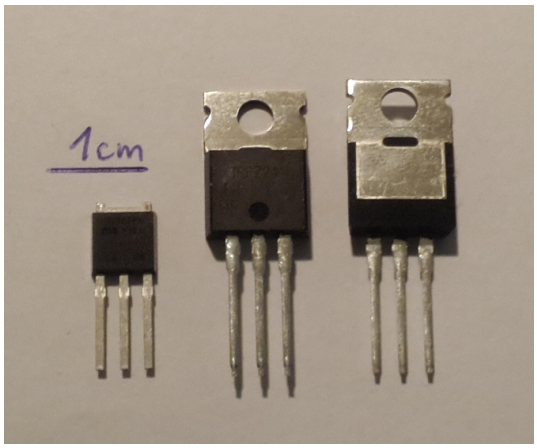


Figure A.1.: The MOSFETs (IRFU 9024N left, IRFZ 24N middle and right).

The left one is the front view of the IRFU 9024N and the middle one is the IRFZ 24N. On the right side is the IRFZ 24N again on the back side. If you look at the left or middle one, the pins from the left to the right are: source, gate, drain. The source is where the signal that should be controlled is connected, in this case the batteries. The drain is the output of the transistor, usually implicitly connected with the device that should be controlled. The gate affects the current flow between source and drain and thereby the current flowing to the device.

A N-Cannel transistor works in the following manner: If the voltage at the gate is below the transistor-specific threshold voltage, the transistor will cut off, otherwise it connects through. Unfortunately there is no perfect threshold voltage and no MOSFET will cut off entirely.

I tested the circuit mainly with the IRFZ 24N which is a N-Channel MOSFET. I tried multiple configurations, for instance the source connected to GND and drain connected to a motor of the car. With GND at the gate, the MOSFET connected through correctly, but the MOSFET burnt through often. I think that is caused by the current peak when the motor starts. Therefore I successfully used a resistor between drain and motor.

Unfortunately there also was a bigger problem. The cutoff worked well if the gate was connected to V_{CC} which, in this case, is delivered by the batteries and has 6V. That is the same potential that is between source and drain because the motor is connected to V_{CC} as well. If I used the high signal of the Raspberry Pi instead, the MOSFET did not cut off properly. There was still a leak flow that was strong enough to run the motor very weak. The high signal of the Raspberry Pi should reach 3.3V but it had only about 2.8V. I thought this should be enough because the MOSFET has a gate threshold voltage between 2.0 and 4.0V. However, since it is not working a higher voltage is required.

As one of my last experiments I tried to use a voltage regulator between the Raspberry Pi GPIO signal and the MOSFET gate. The cutoff was better but still not good enough. Finally I quit this idea and searched for a new attempt.

electrical characteristics at specified virtual junction temperature, V_I = 10 V, I_O = 500 mA (unless otherwise noted)

PARAMETER	TEST CONDITIONS	T_J†	µA7805C			UNIT
			MIN	TYP	MAX	
Output voltage	I_O = 5 mA to 1 A, V_I = 7 V to 20 V, $P_D \le$ 15 W	25°C	4.8	5	5.2	V
		0°C to 125°C	4.75		5.25	
Input voltage regulation	V_I = 7 V to 25 V	25°C		3	100	mV
	V_I = 8 V to 12 V			1	50	
Ripple rejection	V_I = 8 V to 18 V, f = 120 Hz	0°C to 125°C	62	78		dB
Output voltage regulation	I_O = 5 mA to 1.5 A	25°C		15	100	mV
	I_O = 250 mA to 750 mA			5	50	
Output resistance	f = 1 kHz	0°C to 125°C		0.017		Ω
Temperature coefficient of output voltage	I_O = 5 mA	0°C to 125°C		−1.1		mV/°C
Output noise voltage	f = 10 Hz to 100 kHz	25°C		40		µV
Dropout voltage	I_O = 1 A	25°C		2		V
Bias current		25°C		4.2	8	mA
Bias current change	V_I = 7 V to 25 V	0°C to 125°C			1.3	mA
	I_O = 5 mA to 1 A				0.5	
Short-circuit output current		25°C		750		mA
Peak output current		25°C		2.2		A

† Pulse-testing techniques maintain the junction temperature as close to the ambient temperature as possible. Thermal effects must be taken into account separately. All characteristics are measured with a 0.33-µF capacitor across the input and a 0.1-µF capacitor across the output.

Figure A.2.: Snippet of the 7805 voltage regulator data sheet by Texas Instruments.

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 5 sec.)	260°C
ESD Rating (Note 2)	±2 kV
Power Dissipation(Note 3)	Internally Limited
Input Pin Voltage (Survival)	−0.3V to +6.0V
Enable Pin Voltage (Survival)	−0.3V to +6.0V
Output Pin Voltage (Survival)	−0.3V to +6.0V
I_{OUT} (Survival)	Internally Limited

Operating Ratings (Note 1)

Input Supply Voltage	2.7V to 5.5V
Enable Input Voltage	0.0V to 5.5V
Output Current (DC)	0 to 1.5A
Junction Temperature(Note 3)	−40°C to +125°C
V_{OUT}	0.6V to 5V

Figure A.3.: Snippet of the LP38500 voltage regulator data sheet by National Semiconductor.

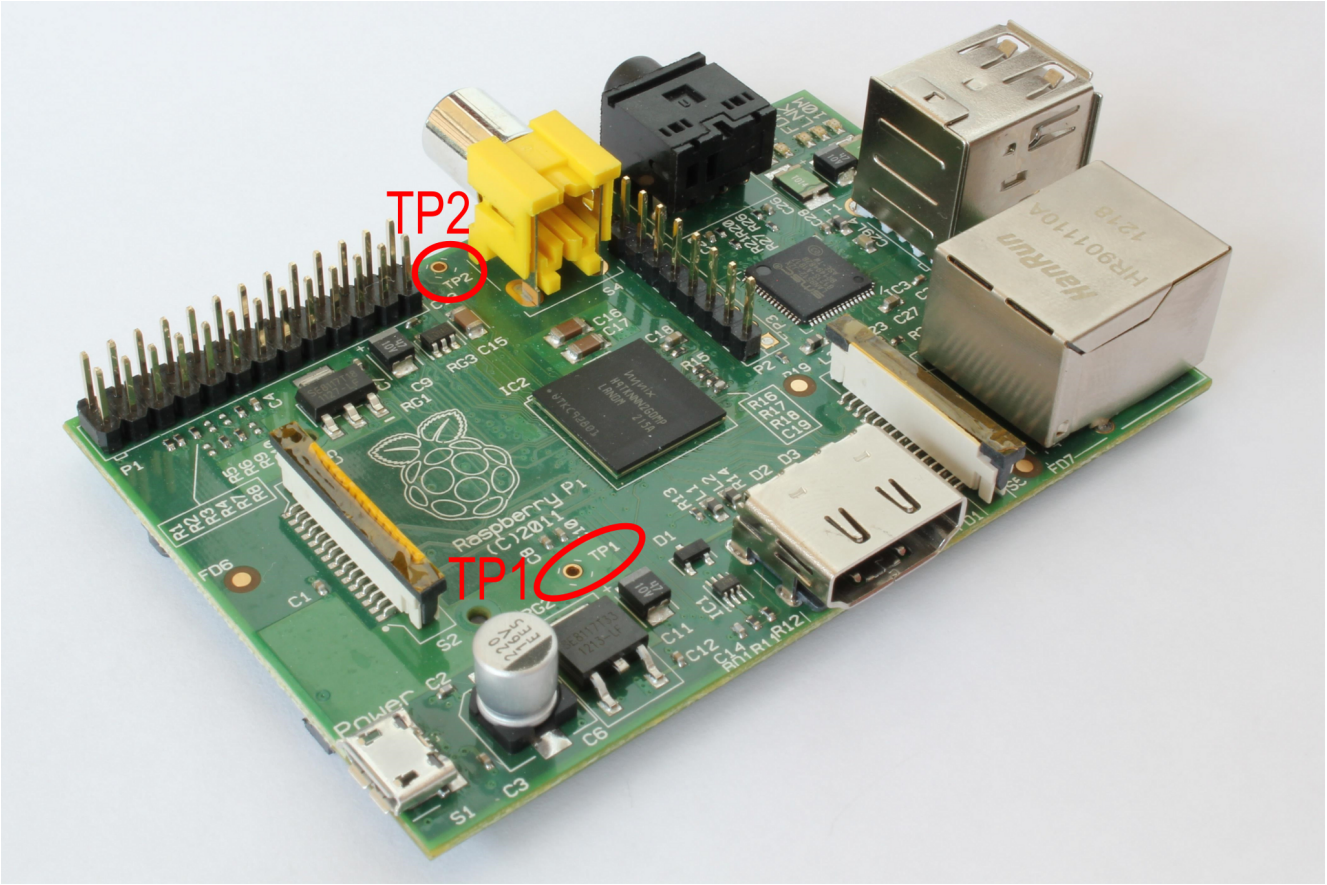


Figure A.4.: Power supply diagnostics points (TP1 and TP2) on the Raspberry Pi.

A.2 Object detection

```
1  LICENSE CONDITIONS
2
3  Copyright (2006): ETH Zurich, Switzerland
4  Katholieke Universiteit Leuven, Belgium
5  All rights reserved.
6
7  For details, see the paper:
8  Herbert Bay, Tinne Tuytelaars, Luc Van Gool,
9   "SURF: Speeded Up Robust Features"
10 Proceedings of the ninth European Conference on Computer Vision, May 2006
11
12 Permission to use, copy, modify, and distribute this software and
13 its documentation for educational, research, and non-commercial
14 purposes, without fee and without a signed licensing agreement, is
15 hereby granted, provided that the above copyright notice and this
16 paragraph appear in all copies modifications, and distributions.
17
18 Any commercial use or any redistribution of this software
19 requires a license from one of the above mentioned establishments.
20
21 For further details, contact Andreas Ess (aess@vision.ee.ethz.ch).
```

Figure A.5.: License of the SURF algorithm.

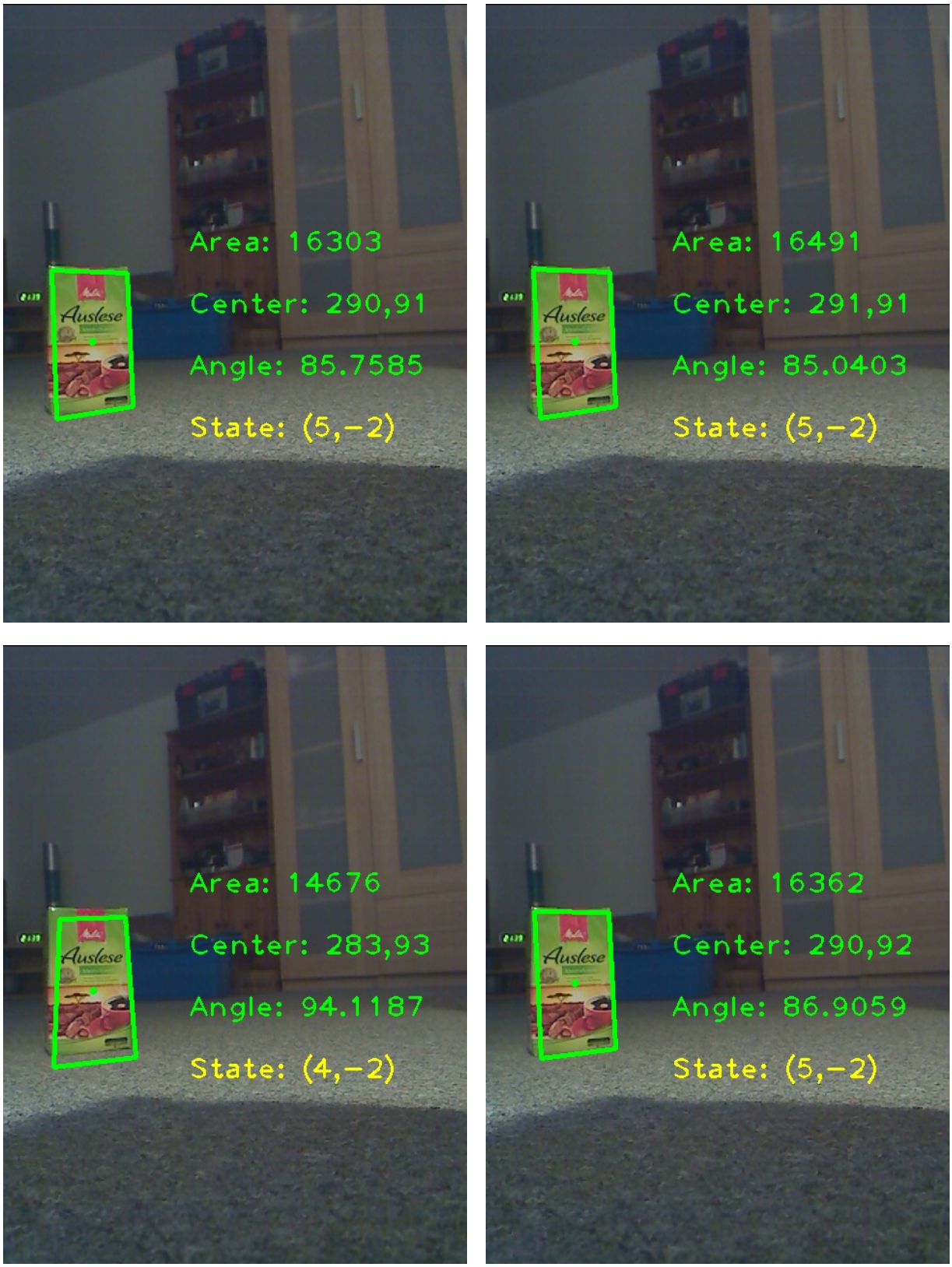


Figure A.6.: Object detection examples. nothing was changed while those images were taken but the detection result still differs caused by noise.

A.2.2 Smoothing of noisy images

To minimize those variations I tried to blur the scene image or use a bilateral filter. A bilateral filter is similar to simple image blurring but tries to preserve edges within the image. Especially the bilateral filter consumes a lot of CPU time and makes the object detection slower. Both methods did not improve the detection quality significantly. Therefore I do not use those filtering methods.

A.3 Class Diagram

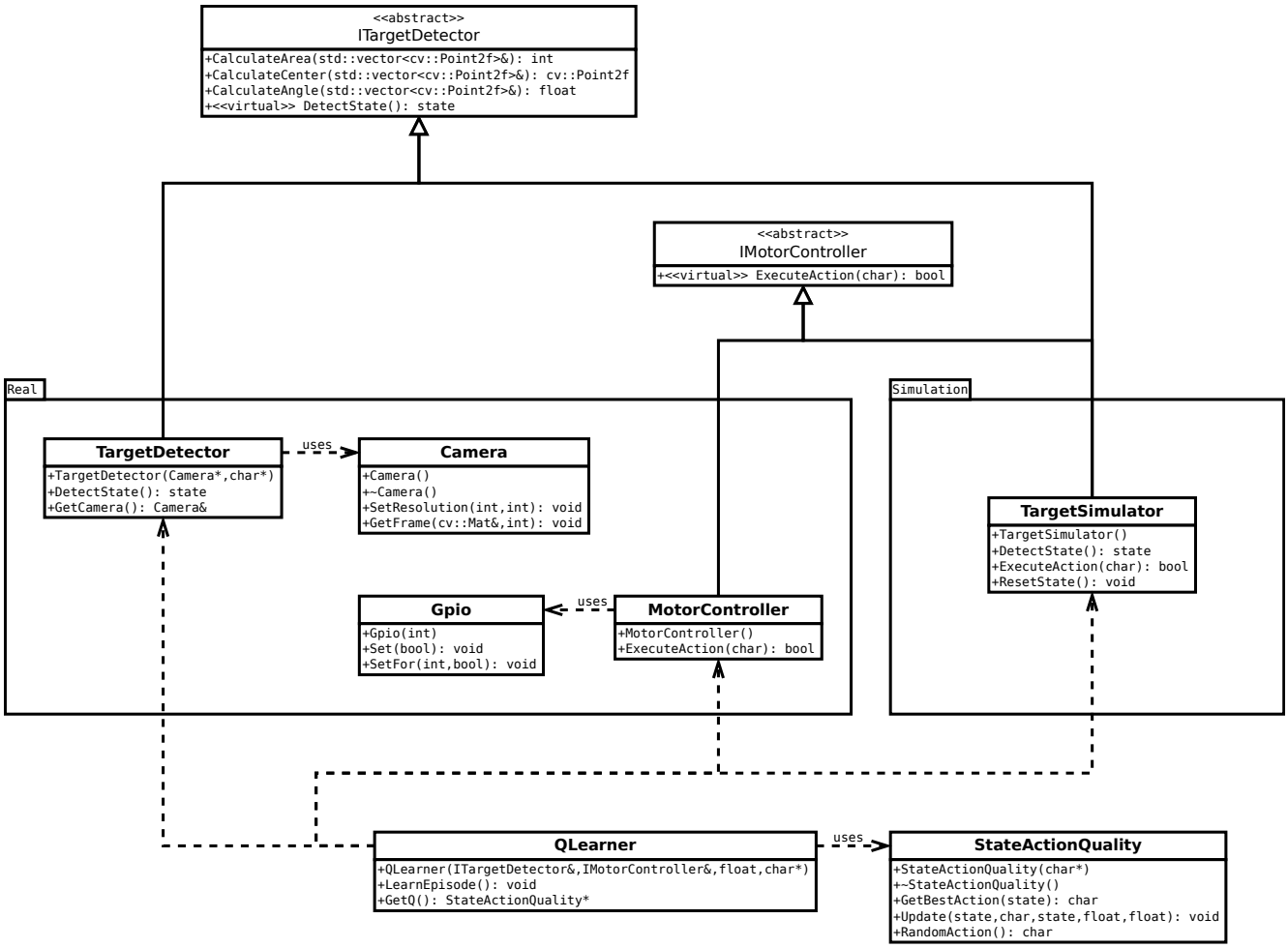


Figure A.7.: Complete class diagram.

Bibliography

Bay, H. et al. (2008). “SURF: Speeded Up Robust Features”. In: *Computer Vision and Image Understanding (CVIU)* 110.3, pp. 346–359.

Juan, L. and O. Gwun (2010). “A Comparison of SIFT, PCA-SIFT and SURF”. In: *International Journal of Image Processing (IJIP)* 3.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Watkins, C. (1989). “Learning from Delayed Rewards”. PhD thesis. Cambridge University, Psychology Dept., Cambridge, UK.