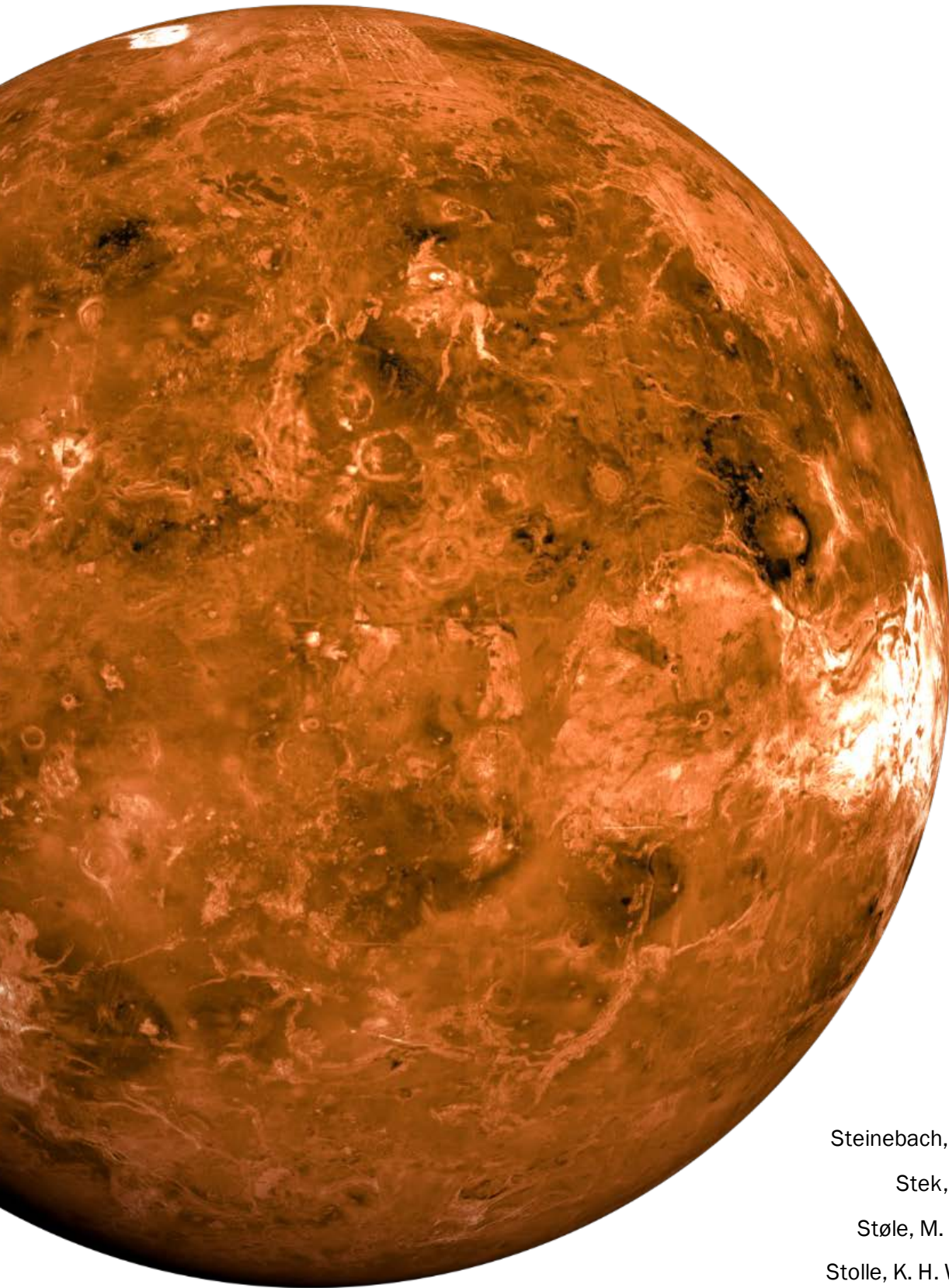


Autonomous Venus Exploration and Sample Collection Robot

5XIB0 Final Report



Group 32

Tutor:

Avan, D. B.

Participants:

Steinebach, L. [0947022]

Stek, T. [0957900]

Støle, M. B. [0980907]

Stolle, K. H. W. [0945431]

Timmerman, N. A. G. [0962892]


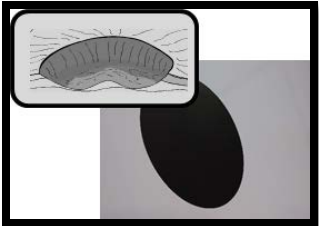
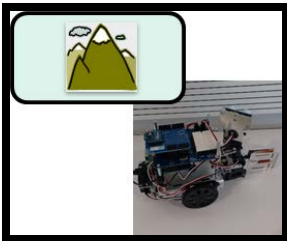


Contents

Contents.....	i
1 Introduction.....	1
2 System-level design.....	2
3 Component-level design	4
3.1 Microcontroller	4
3.2 Actuators.....	4
3.3 Sensors.....	4
3.5 Research lab.....	7
4 Strategy and algorithm	8
4.1 Algorithm structure and basics.....	8
4.2 Mountain-navigation algorithm	10
4.3 Sweep-navigation algorithm.....	13
5 Testing	16
5.1 Infrared Sensors.....	16
5.2 Ultrasonic sensors.....	17
5.3 Grabbing Samples	17
5.4 Returning to the lab.....	18
5.5 Navigation	18
6 Conclusion.....	19
7 Discussion.....	19
7.1 General.....	19
7.2 Software.....	19
7.3 Hardware.....	19
8 Contributions.....	20
8.1 Final report.....	20
8.2 General.....	20

1 Introduction

One can imagine that research on the planet Venus is not a simple task. The planet has a lot of natural obstacles, such as its high volcanic activity and its atmosphere (which has a relatively low oxygen concentration). Therefore, human exploration is unlikely. Robots, however, may be a good option to carry out this research and exploration, since they can be made much more resistant to natural effects than humans.

To test the viability of this idea, our group was asked to design two robots which can execute several tasks on a simplified model of the planet. The robots, equipped with various sensors and actuators, must be designed to drive around on the planet and find as much research samples as possible in minimal time. While the robot is driving it should avoid natural obstacles such as cliffs and hills. A basic set of components and a robot skeleton was supplied to kick start the project. The details of the simplified model where the robots will be tested on are shown below.

Object	Known properties	
Boundary (black tape)	Absorbs infrared light.	
Cliff (black tape)	Absorbs infrared light.	
Hill	Reflects ultrasound. Absorbs infrared light. Minimal height: 30 cm.	
Rock sample (white cardboard)	Reflects infrared light. Dimensions: 2 x 2 x 2 cm.	
Lab (wooden box with ramp)	Height: 2.5 cm. Area: 20 x 20 cm.	

Translating the problem in terms of this material model, the final assignment can be described as follows:

Design and program two robots, which can efficiently search for research samples and drop these off at the specified research lab. The robots will have to avoid the hills, cliffs and boundary.

2 System-level design

The provided robot is based on the 'Parallax Shield Kit', which contains a basic metal construction, an Arduino controller, a gripper and powered wheels. Additionally, the robot was supplied with an ultrasonic distance sensor, a breadboard, a wireless communication module (ZigBee) and digital encoders on both wheels.

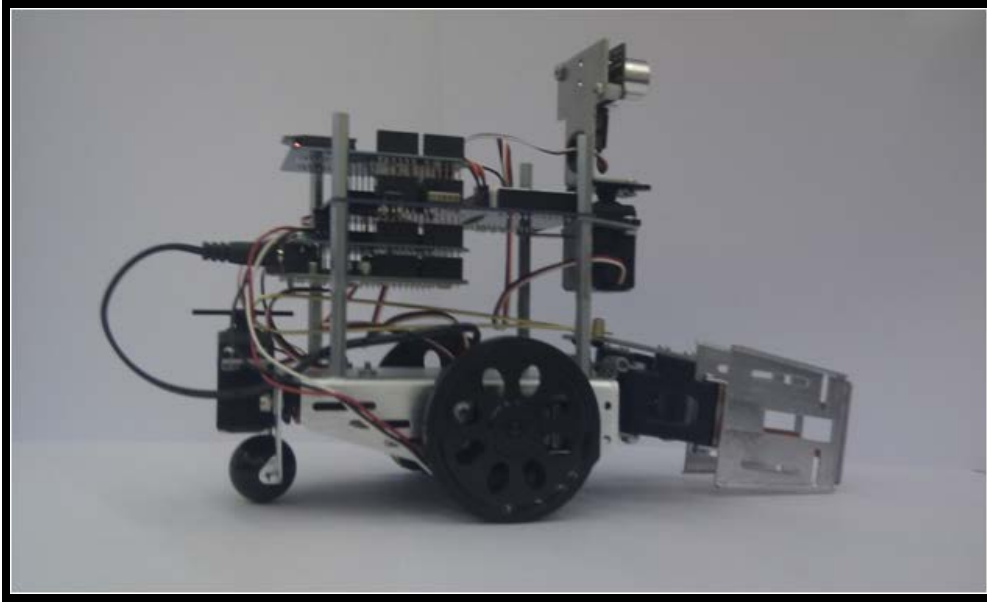


Figure 1: The robot

In order to be able to successfully execute the strategy described above, the robot must be able to do each of the following tasks:

- Drive around.
- Pick up and drop research samples.
- Communicate with the other robot.
- Detect hills, cliffs, research samples, the research lab and the boundary of the given area.

From hardware perspective the first, second and third conditions are easily fulfilled, since the provided 'barebones' robot contains powered wheels, a gripper, and a communication module. For the last condition though, some additional sensors may be needed. The hills can easily be detected by the ultrasound distance sensor. This cannot be used for the cliffs, samples, lab and the boundary though, because these do not have a sufficient height. The only known property of these objects which can be used to detect and distinguish them is the infrared reflectivity. Therefore, it was decided to add multiple infrared reflection sensors to the robot (marked in red in the picture, to distinguish it from the components included in the basic kit). It was also decided to attach an additional ultrasound sensor on the bottom of the robot to be able to detect samples from a long distance. The final robot and its core components are summarized in the graph below.

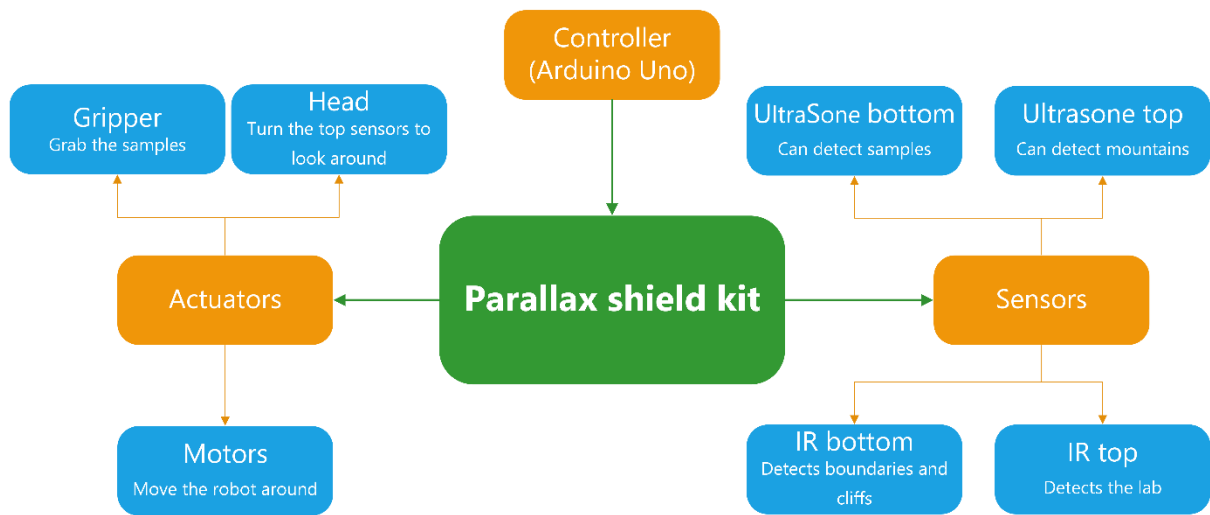


Figure 2: System level description of the robot

3 Component-level design

Below the individual components will be described in more detail. At the end of this chapter there is an extensive list of all the components, their serial numbers, the pins (on the microcontroller) that they use and their placement on the robot.

3.1 Microcontroller

The robot has an Arduino Uno (ATMega38P-based) microcontroller at its core. The controller runs at an operating voltage of 5V, has 14 digital I/O pins, 6 PWM-digital I/O pins, 6 analogue input pins and is based on the AVR-architecture.

The Arduino can be programmed using the 'Arduino Language', a language that is transpiled to C and then compiled using AVR-GCC (a version of GCC specific for the AVR-architecture). The language is essentially a subset of C that hides some of the less intuitive parts of the AVR architecture from the programmer. The Uno does unfortunately not allow for concurrent programming, but it does have a system in place for software interrupts, allowing the programmer to schedule periodic tasks (e.g. checking the input of a sensor).

The Arduino will handle all software logic in the robot.

3.2 Actuators

3.2.1 Wheels

The robot has two wheels, each driven by a separate continuous-rotation servo (Parallax #900-00008). The speed at which a wheel turns can be modulated using a PWM-signal. Below the maximum speed, which is about 50 RPM, it is not possible to directly control the speed at which the wheel is turning. This would require some feedback or tracking system. This feedback is available at the robot through the digital encoders on the wheels, see *chapter 3.3*. The signal sent to the wheel will only control its acceleration.

The wheels are connected to the controller via PWM pins 12 and 13.

3.2.2 Additional servos

The robot has two basic rotation servos (Parallax #900-00005): one to control the grabber and one on top of the robot to turn the ultrasonic sensor left and right. This type of servo is an electronic motor that uses a mechanism to provide itself with negative feedback, allowing for very precise control of the angle made by the motor. This angle is, again, dictated by a PWM-signal. Some rubber is mounted on the inside of the grabbing device to achieve better grip while carrying samples.

The grabbing device is used to pick up samples, carry them to the lab and to drop them off. It is connected to PWM pin 10 on the controller. The servo for the ultrasound sensor is connected to PWM pin 11.

3.3 Sensors

3.3.1 Ultrasonic sensors

An ultrasonic sensor will check the time it takes for a self-emitted ultrasound wave to reflect back to itself. The difference between the point in time when a sound was emitted and the point in time when the sound was reflected and detected allows for calculation of the distance the sound has travelled.

Each of the robots have two of these sensors installed, a 'Parallax 28015 REV C' at the top and a 'HC-SR04' at the bottom. It was decided to order this additional HC-SR04 sensor because it has a

similar range to the already present sensor, about three meters. This is really useful to distinguish samples from mountains at a very long distance. There is unfortunately no possibility of tilting the top sensor at a downwards angle. This means that this sensor can only be used to detect objects that are tall enough to 'catch' the ultrasound emitted by the sensor.

The sensor at the top is used to detect mountains and help navigate, while the one mounted underneath the robot is used to detect samples. Their echo wires are connected to digital pins 2 and 3 respectively.

3.3.2 Digital encoders

Two digital encoders on both wheels are used to give feedback to the controller about the RPM of the wheels. The wheels have got 8 holes and spokes. Infrared sensors in the encoders can detect the spokes of the wheel. When the sensor detects a hole, it will have a high output. And when a spoke is detected, the output will be low. In this way the controller can, based on the encoder output, calculate how fast the wheel is turning. This will be helpful to perform special manoeuvres or when a certain distance should be covered.

The left encoder is connected to digital pin 7 on the Arduino and the right to pin 8.

3.3.3 Infrared sensors

Infrared light is a type of light with a frequency close to red light, but outside of the visible spectrum. An IR-sensor will measure the intensity of red light – from all directions – cast upon it. The effective range (i.e. the range from which IR-light may be emitted for a significant measurement) is 100cm for detection purposes and 20cm for analysis purposes.

Each robot has two IR-sensors mounted underneath, one in front of each wheel. These are needed to detect lines and cliffs and thus only need to have a short range. The CNY70 reflective sensor was chosen for this purpose. The CNY70 has a IR-LED and a phototransistor built-in. The phototransistor is sensitive for infrared light. A circuit was designed and built to be able to connect it correctly to the controller. An analogue output is chosen, because the robot should distinguish white, grey and black. Figure 3 shows the design of this circuit.

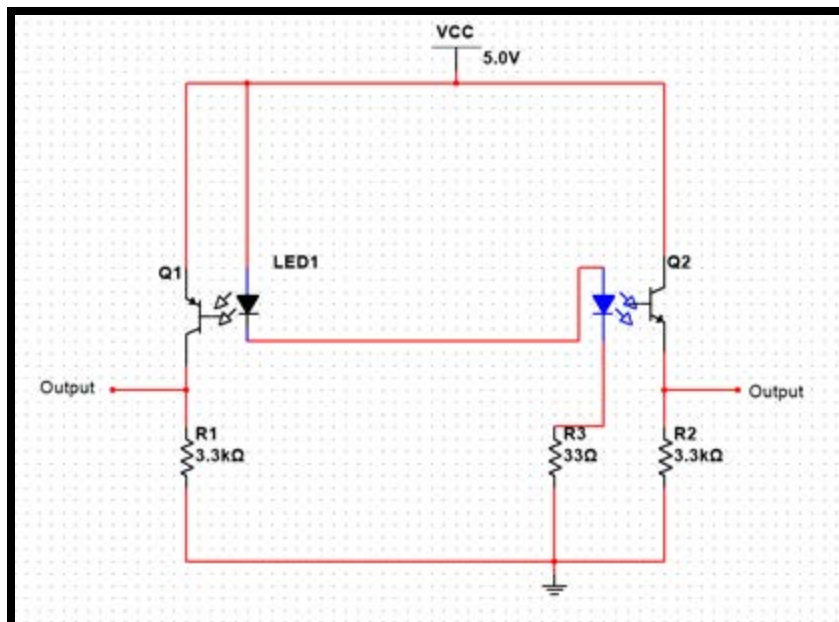


Figure 3: Circuit of IR-sensors underneath robot

It was decided to add two additional sensors to the robot. One to detect real samples, which reflects infrared. The other one to detect the research lab, which has an infrared beacon installed. Therefore, the robots have an extra IR-sensor attached to their top ultrasonic sensor and one to their grabber. One of these is a line-tracking module ordered from dealextreme.com,

Figure 4 shows the design of the ordered sensor, figure 5 shows the design of the replica. As a result of limited time the replica was not included in the final system design.

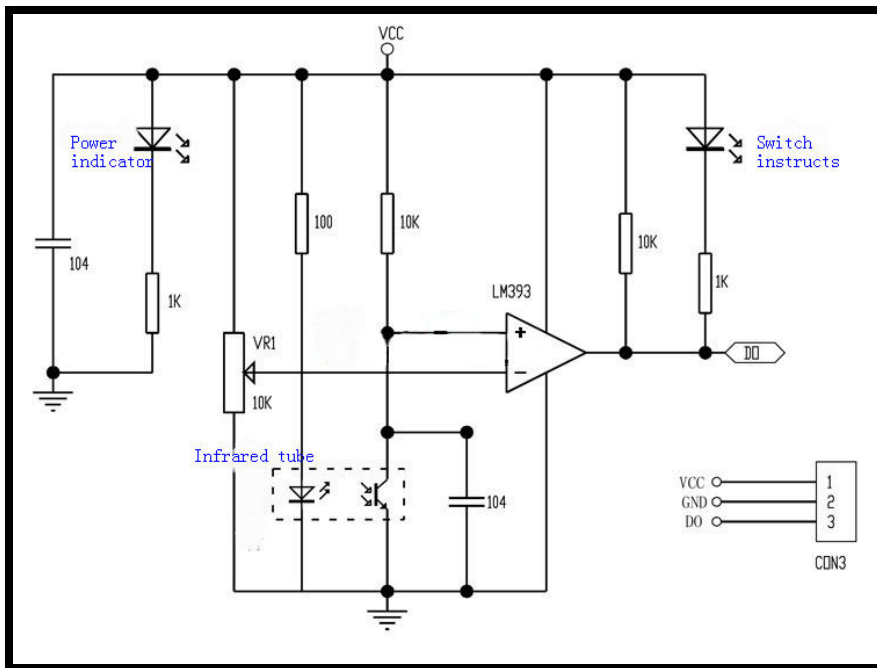


Figure 4: Schematic of ordered IR-sensor

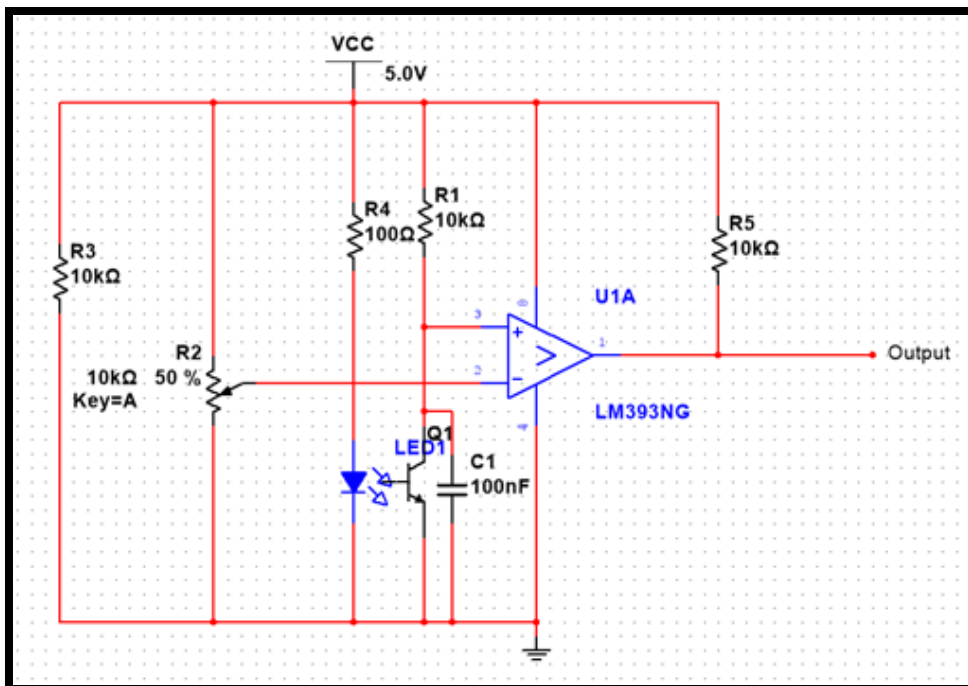


Figure 5: Schematic of replica sensor

The left and right IR-sensors are attached to analogue pins 0 and 1, respectively. The bottom and top ones are attached to digital pins 4 and 5.

3.5 Research lab

The lab consists of four pieces of wood made into a square. On one side of the lab there is a ramp which the robot needs to drive up to be able to drop samples into the lab. Some modifications were made to the lab in order to make it easier for the robot to get to the lab and up the ramp. The ramp was masked with white paper and black tape so the robot can detect the edges of the ramp. Another modification is the addition of a small tower. On top of this tower a beacon of infrared emitters is mounted, so the robots can detect the lab. Eleven infrared LEDs were used to emit a constant infrared light. The robots infrared transistors are mounted at the same height to detect at which angle the robot is currently standing with respect to the beacon. The transistors are attached on the ultrasonic sensor so they automatically sweep as well.

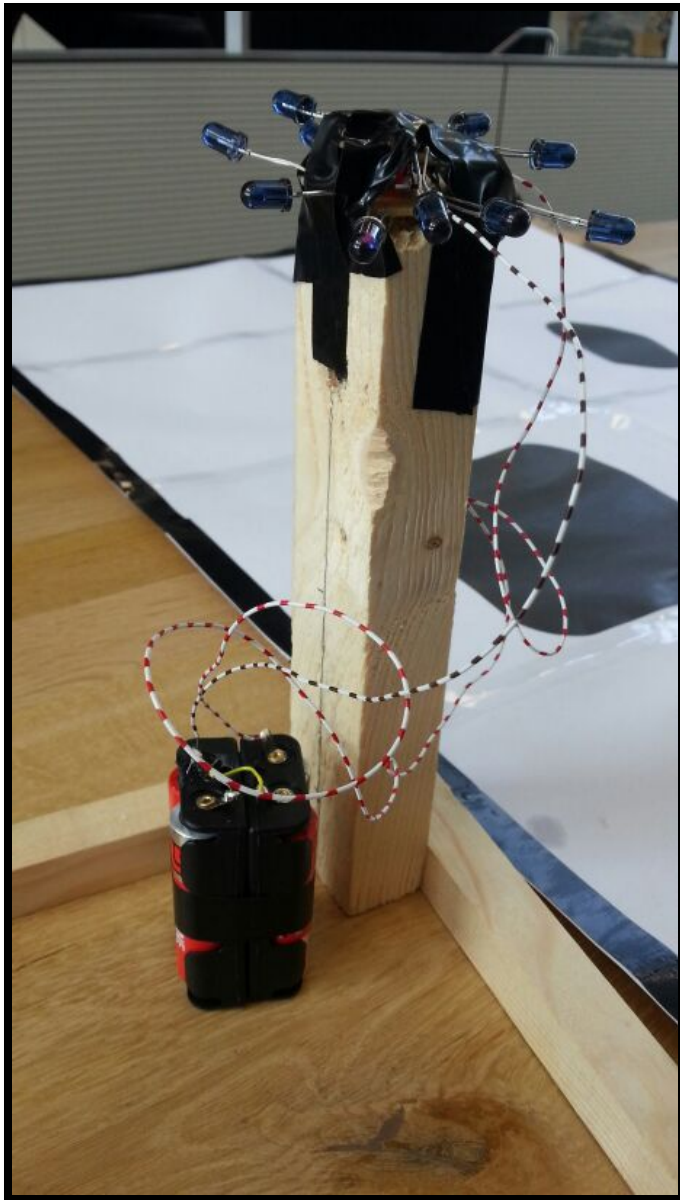


Figure 6: The IR beacon on the lab

Component	Pin on controller	Placement on robot	Part number
Microcontroller (Arduino Uno)	-	Middle	-
Servo wheel left	12 [Digital PWM]	Bottom left	Parallax #900-00008
Servo wheel right	13 [Digital PWM]	Bottom right	Parallax #900-00008
Servo ultrasonic top	11 [Digital PWM]	Top	Parallax #900-00005
Servo grabber	10 [Digital PWM]	Bottom	Parallax #900-00005
Ultrasonic top	Trigger: 9 [Digital PWM] Echo: 2 [Digital]	Top	Parallax Ping)))
Ultrasonic bottom	Trigger: 6 [Digital PWM] Echo: 3 [Digital PWM]	Bottom	HC-SR04
Left encoder	7 [Digital]	Left behind wheel	
Right encoder	8 [Digital]	Right behind wheel	
Infrared top	5 [Digital PWM]	Top	Unknown (dx.com SKU: 213600)
Infrared bottom	4 [Digital]	Bottom	Replica of the above
Infrared left	0 [Analog]	Bottom left	CNY70
Infrared right	1 [Analog]	Bottom right	CNY70
Wireless module	RX: 0 [Digital] TX: 1 [Digital]	Top	XBEE 802.15.4

4 Strategy and algorithm

Two algorithms have been developed, one for each robot: an algorithm which uses mountains as a reference to navigate and an algorithm which lets the robot sweep over the field. Before these algorithms are detailed described, the basic structure and common code which these algorithms use will be described.

4.1 Algorithm structure and basics

4.1.1 Object-oriented approach

An object-oriented approach was chosen, since this allows for modular design and easy expansion of the total algorithm. Both of the algorithm variants are described in a separate class and use a set of common classes which abstract the interface with the hardware. The most important one of these is the 'Robot Controller' class, acting as the main interface to the robot's sensors and actuators. Additionally, classes are defined for sensors and actuators ('Infrared', 'Ultrasonic' and 'Servo').

The 'main.ino' file acts as the entry point for the program and instantiates the robot object and its sensors, the interrupt routines (to be discussed in the next section) and the algorithm variant itself. Pre-processor directives are extensively used to define constants for things such as pin numbers and the algorithm variant.

4.1.2 Interrupt Service Routines

In order to be able to constantly check whether the robot drives over a cliff or approaches a mountain, the robot's sensor values must be continuously updated. Since manual updating (by checking for the time) in the algorithm loop is often subject to errors, it was decided to make use of software interrupts, also known as ISR's (Interrupt Service Routines).

The Arduino IDE does not include a native library to easily implement these interrupts. Therefore, the library 'Arduino Thread' by Ivan Seidel was implemented. This library provides a basic thread

controller. This controller can be assigned tasks (functions), for which a specific time interval can be specified. When the controller is called, it will decide if a task needs to be called (based on the elapsed time since the last time it was called).

Two tasks were assigned to the controller: one which updates the robot movement and one which makes the robot update its sensor values. Both of these tasks were set to an interval of 25 milliseconds. Using the 'Timer One' library, a timer was added in order to interrupt the program every 25 milliseconds to call the thread controller.

4.1.3 Movement update routine

When this routine is called, the robot object will be asked to update its movement. The object will then do the following things:

- *Determine the current RPM of the wheels*
The sensors next to the wheels of the robot output a signal which represents the turning speed of the wheels. The rising and falling edges of this signal are used to determine PWM intervals. Using these intervals, the RPM of the wheels can be calculated quite accurately.
- *Determine the amount that the robot has turned*
The robot has an internal turn target variable, which tells the robot how much degrees it has to turn in a clockwise direction. Using the calculated RPM of its wheels, the robot determines how much distance each wheel has travelled. When these distances are not equal to each other, the robot will recognize that it must have been turning, and remove the amount of degrees that it has turned from its target.
- *Update wheel velocities*
The robot's internal speed variable can be set using various methods ('Forward', 'Reverse') in the robot controller class. However, to determine the speed of its wheels, the robot will first check its turn target. When this is a big value, the robot will set the wheel speed to the value that has been set, but it will make the wheels move in opposite directions. When this is a small value (< 15.0 degrees), the robot will set the wheels to turn at a quarter of the speed that has been set, to allow for more accurate turning. When the turn target is negligible, the robot will completely remove it and will just set the wheel speed to its internal speed variable.

4.1.4 Sensor update routine

When this routine is called, the robot object's scan method will be called. The robot will then emit a sound pulse from each of its ultrasonic distance sensors. Moreover, when the robot detects that it was still waiting for a response from one of those sensors, it will assume that the previously sent pulse was absorbed by something or that the object was too far away. Consequently, it will set its internal distance variable to `DISTANCE_INFINITE` (a defined constant) to indicate this.

Unfortunately, the thread interval of 25ms is way too long to catch an ultrasonic sensor response (which takes a few milliseconds at max). Simply waiting for a response after sending a pulse is also not an option, as this is devastating for ISR timing. Luckily, besides the interrupts called by the thread controller, the native pin interrupt functionality of the Arduino can be used to catch the response from the ultrasonic sensors. The Arduino Uno only has two pins which have this interrupt functionality, just enough for our two ultrasonic sensors.

The native pin interrupts were set to trigger when the echo pin of an ultrasonic sensor changed its state. When this interrupt executes, the robot controller is called to determine the measured distance based on the time between the start and the end of the response pulse. It will then update its internal distance variables with the new values.

4.1.5 Avoid procedure

The procedure to avoid (drive around) cliffs is a general action that the robot will have to perform, and is therefore included in the parent class from which both algorithms are derived. The operations executed in this procedure are listed below in chronological order:

- *Register the side at which a cliff is detected*
The robot uses its left and right infrared sensors to detect the angle at which it is driving into the boundary or cliff. When it only detects a black colour with its left sensor, it will remember this and try to turn to the right later, and vice versa. When both sensors detect black, the robot will default to the right.
- *Drive backwards for about half a second*
This provides the robot with some extra space for turning later in the procedure.
- *Try to find a passage*
The robot will try to turn in the direction that it has determined. Then it will drive forward for a couple hundred milliseconds and turn back to 'look' at the line/cliff again. It will drive forward to test if it has found a passage. If it does, it will try to compensate for the distance it has travelled in the left or right direction by turning again and driving until it arrives at the path it was traveling before. If no passage is found, the robot will try again to find one two more times. If after this the robot still has not found a passage, it will give up and it will return a 'false' to indicate that it did not succeed and that it probably encountered the boundary of the field.

4.2 Mountain-navigation algorithm

This algorithm uses the fact that the robot can detect mountains and samples from a long distance using its ultrasonic sensors. The basic way it works is very intuitive, and can thus easily be understood: The robot looks around using its two ultrasonic sensors. When it sees a sample, it will drive straight toward it. When it cannot see a sample, it will assume that a mountain is blocking its view and it will try to drive behind the mountain and scan again. This has been split up in several procedures, each of which will be covered below. At the end a graph, which summarizes the whole algorithm, is included.

4.2.1 Sweep procedure

This is the procedure that the robot will start in, when it is placed on the field and reset. It will fix its top ultrasonic sensor to a forward position and do a couple of tasks. Note that there can be sudden jumps to other procedures and therefore not all tasks are always executed. This is described in more detail below.

- *Scan for samples*
The robot will turn 90 degrees to the left, while continuously checking the distance values coming in from its ultrasonic sensors (which have a very long range). If the value it measures using the top sensor is more than 10 centimetres bigger than the value it measures using its bottom sensor, it will assume that this means that it is directed at a sample. It will then stop turning, staying pointed in the direction of the sample, and jump to the sample procedure. If it does not detect a sample, it will turn 180 degrees to the right and try again.
- *Scan for mountains*
After the robot has had no luck with detecting samples at a long range, it will try to find a mountain to drive to. This works in the same way as the detection of samples, except for the fact that it now assumes it has found a mountain when both of the sensors give the same value (within 10 centimetres). When a mountain is found, the program will jump to the mountain procedure.
- *Assume mountain*
If the robot arrives here, it clearly has not found anything to drive to. It will now assume that it is directed at a mountain and will jump to the mountain procedure as a last resort.

4.2.2 Sample procedure

If the algorithm arrives at this procedure, the robot must have detected a sample using its ultrasonic sensors and it must point directly in the direction of it (see the section on the sweep procedure). It will now try to drive towards the sample and pick it up, by means of the following steps:

- *Loop while the border is not reached*
If the robot has not yet arrived at the border, it will drive forward and constantly check the following:
 - Mountains
If the top ultrasonic sensor returns a distance of less than 25 centimetres, the robot is likely to be very close to a mountain. Since it entered this procedure under the assumption that it was directed at a sample, this is unexpected (it is possible that there is a sample in front of the mountain, but it would have detected that earlier using the third check below). Therefore, the robot will recognize that its assumption to be directed at a sample is wrong and it will jump to the mountain procedure instead.
 - Cliffs / boundary
When the left or right IR-sensors sense a black colour, the robot will call the avoid method to try to drive around it. If this does not succeed, the robot might have reached the border and it will exit the loop.
 - Samples
If the bottom ultrasonic sensor of the robot returns a value of less than 20 centimetres, the robot probably stands in front of a research sample. It will drive until the sample is within grabbing distance, grab it, and jump to the lab procedure.
- *Give up*
If the robot arrives here (because it has reached the border), it will give up searching for the sample. It will turn 180 degrees and jump to the mountain procedure.

4.2.3 Mountain procedure

If the algorithm arrives at this procedure, the robot must have detected a mountain using its ultrasonic sensors and it must point directly in the direction of it (see the section on the sweep procedure). It will enable the turning of its top ultrasonic sensor and it will try to drive towards the mountain, in a way similar to the method it uses to arrive at a sample:

- *Loop while the border is not reached*
If the robot has not yet arrived at the border, it will drive forward and constantly check the following:
 - Mountains
If the ultrasonic sensor on top of the robot returns a value of less than 25 centimetres, a mountain is somewhere around the robot (because its top sensor is constantly turning). The robot will try to direct itself in the direction that its top sensor is pointing at, and it will disable the constant turning of it. Now it will try to drive around the mountain, in a way similar to the avoid procedure. It will try this three times. Regardless of whether it succeeds or not, the robot will assume that it has arrived at a useful position for a new round of sweeping and it will jump to the sweep procedure.
 - Cliffs / boundary
When the left or right IR-sensors detect a black colour, the robot will call the avoid procedure to try to drive around it. If this does not succeed, the robot might have reached the border and it will exit the loop.

- *Give up*

If the robot arrives here (because it has reached the border), it will give up searching for the mountain. It will turn 180 degrees and jump to the sweep procedure.

4.2.4 Lab procedure

If the robot arrives in this procedure (because it thinks it has picked up a sample), it has to return to the lab. Due to some problems during the development of the algorithm in the last week, this part could not be completely finished (see the testing & discussion parts). The code which was actually written for this performed the following actions:

- *Scan for the lab*

The robot uses its top infrared sensor to try and find the lab from a long range. It will enable the turning of the ultrasonic sensor, and if it has detected an infrared signal, it will turn in the direction the sensor is pointing at. It will then disable the turning and start driving to the lab (not implemented). If it has not found the lab yet, it will turn 180 degrees and try again 10 more times. If it does not succeed after this, it will jump back to the sweep procedure.

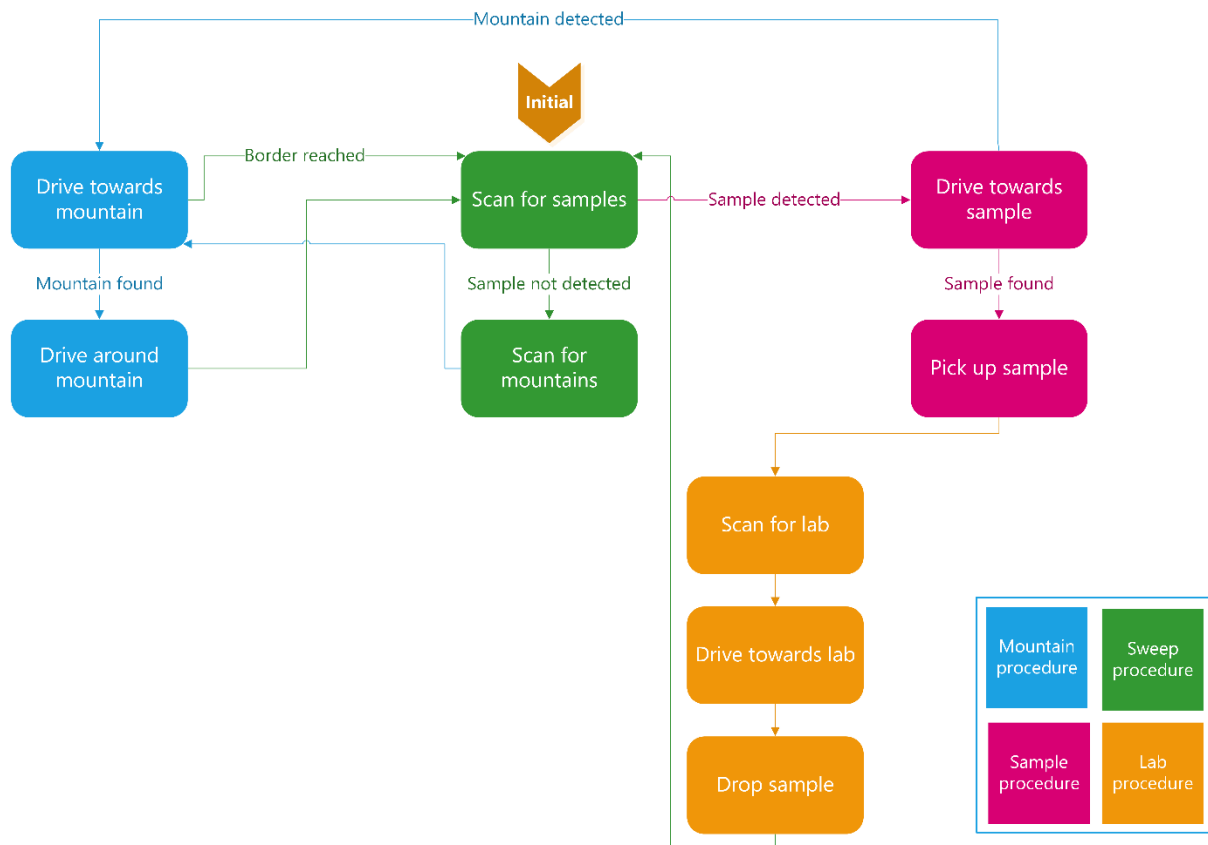


Figure 7: Flowchart mountain algorithm

4.3 Sweep-navigation algorithm

The sweep-navigation algorithm is a simpler and less efficient algorithm than the mountain-navigation algorithm. This algorithm will just let the robot randomly drive over Venus and every time it detects a mountain or cliff, it will just turn around and look further for samples.

4.3.1 Sweep procedure

The sweep procedure is the initial procedure. This function is used to sweep around the field and search for samples. This function will also react when a cliff or mountain is detected near the robot. The strategy of finding the samples in this algorithm is as follows:

- The robot starts moving forward
- The upper ultrasonic sensor will be orientated to the front, and then sweep continuously from -30deg (left) to 30deg (right). If a mountain is detected less than 20 cm from the robot, the algorithm jumps to the Avoid mountain procedure.
- The bottom IR-sensors both check if there is a cliff or border in front of the robot. If a cliff is detected, the algorithm will go the Avoid cliff function. It will also store at which side the cliff is detected.
- If the difference between the measurement of the bottom ultrasonic sensor and the top ultrasonic sensor is bigger than 10 cm, the algorithm will conclude that a sample is detected. And it will jump to the sample procedure.
- If by now the robot's procedure hasn't changed, the sweep procedure function loops.

4.3.2 Sample procedure

If a sample is detected, the robot will first go to the sample, pick it up and then the algorithm will go to the lab procedure.

- The robot drives forward until the distance to the sample is smaller than 20 cm. During this process the robot will still react on cliffs and mountains as described in the sweep procedure.
- If the sample is closer to the robot than 20 cm, the robot won't react to cliffs and mountains anymore. It will go forward, on a slower rate, until the distance from the sensor to the sample is 6 cm.
- On this moment the robot stops and will close its grabber. Now the algorithm will go to the lab procedure.

4.3.3 Avoid mountain procedure

The avoid mountain procedure is used for avoiding mountains. To make sure the whole map is covered, the robot will first look to the left, when there is room at the left of the robot, it will go left. If left is not free, then the robot will look right. If right is free, it will go parallel to its previous driving direction, but in the opposite direction and 10cm shifted to the right. If both sides are not free, the robot moves 20cm back and tries again. The strategy is as follows:

- Turn the Ultrasonic sensor to the left, wait for the sensor thread to fire once, check the measurement variable (updated in the sensor thread). If the way is free then make a 90 degree turn to the left and go back to the Sweep procedure.
- When the left side was not clear, turn the Ultrasonic sensor to the right, wait for the sensor thread to fire once, check the measurement, if the way is free, then move 10 cm to the right, and make another 90 degree turn. The robot will look for samples in the next lane. The Sweep procedure will be called again
- If the algorithm arrives at this point, then both left and right are not free. The robot will move back 20cm, turn around and will go back to the sweep procedure.

If the robot detects a mountain while it is going to a sample, it will go around the mountain instead of turning back. Then the procedure is as follows:

- Turn the ultrasonic sensor to the left, wait for the sensor thread to fire once, check the measurement variable.

- If the way is free then turn 90 degrees to the left and turn the sensor back to the front.
- The robot will drive for 10 cm, where after the robot will make a 90 degree turn to the right to look if it passed the mountain.
 - If this is not the case, the robot will turn back 90 degrees to the left, go another 10 cm forward and make another turn to the right to check if it passed the mountain. The algorithm will try this up to 3 times, if the mountain is not passed after these 3 attempts it will go back to the sweep procedure.
 - If the robot did pass the mountain, it will do the same procedure on the new side of the mountain and eventually go the same distance to the right as it did in the beginning of this procedure.
- If the robot successfully passed the mountain, the algorithm will jump back to the sample procedure.

4.3.4 Avoid cliff procedure

This procedure is very similar to the avoid mountain procedure and will avoid cliffs and borders. Instead of looking to the left and right this function already received at which side the cliff is located. So this function will go directly to the left or turn around.

- When the cliff or border was detected at the left, the robot will turn left and the algorithm will switch back to the sweeping procedure.
- If the cliff or border is discovered at the right side of the robot, the robot will make a 90 degree turn, go 10 cm forward and make another 90 degree turn. The algorithm will return to the sweeping procedure and the next lane will be observed, looking for samples.

If the robot detects a cliff while it is going to a sample, it will go around the cliff instead of turning back. The manoeuvring of this approach is nearly the same as the one of the avoid mountain procedure, while going to a sample.

- If the cliff was detected with the left IR-sensor, the robot will try 3 times to go around the cliff at the right side. If it was detected with the right sensor, it will try to go left.
- The exact procedure, with trying 3 times to go passed the cliff is the same as described in *4.3.3 Avoid mountain procedure*. Only in this case it will of course use the IR-sensors.

4.3.5 Lab procedure

See *4.2.4 Lab procedure*. This procedure is the same as in the Mountain-navigation algorithm.

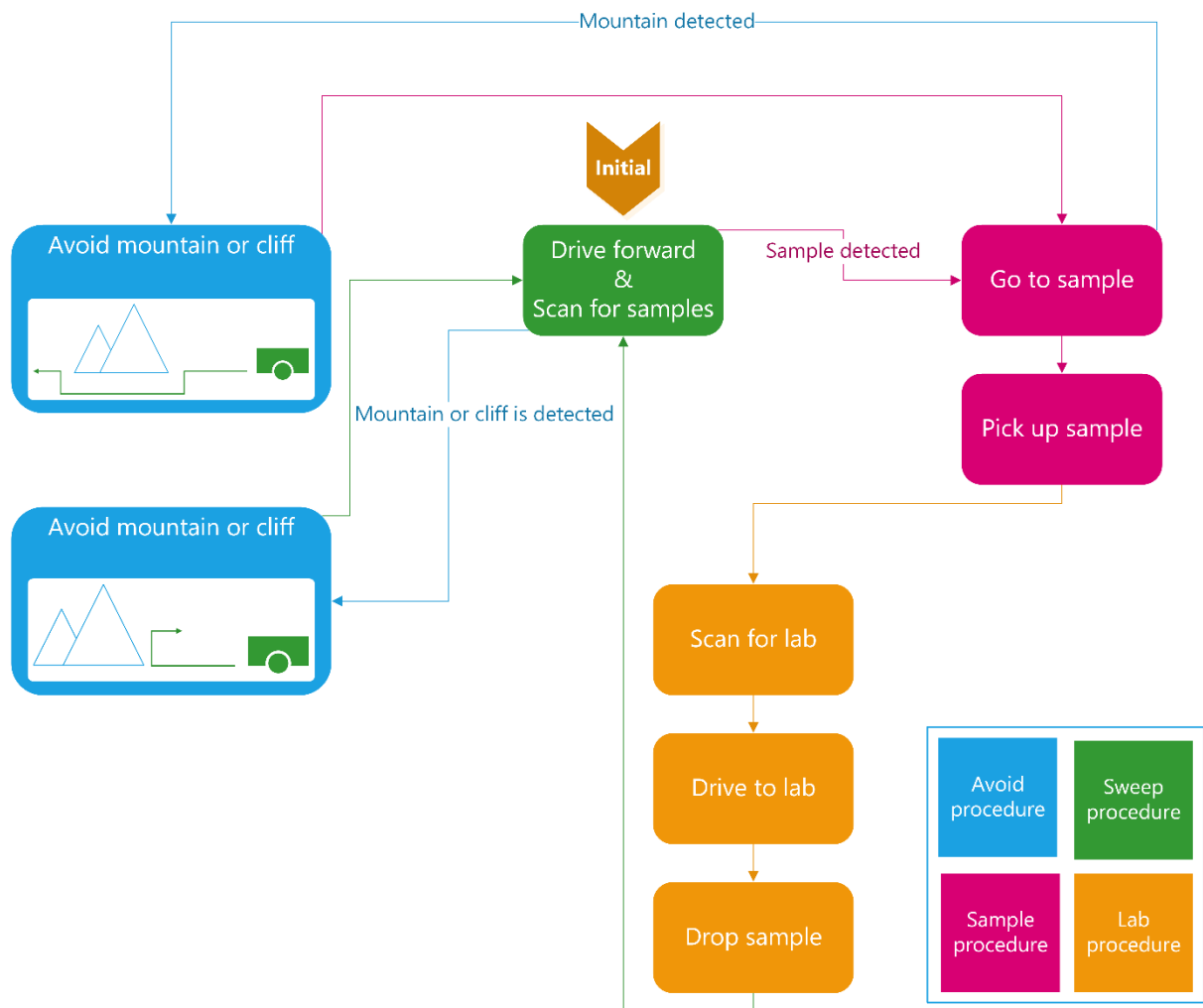


Figure 8: Flowchart sweep algorithm

5 Testing

5.1 Infrared Sensors

5.1.1 Line Detection

After soldering the circuits of the CNY70 the circuits are tested by measuring the voltage at the output, when holding a white campus card in front of the sensor. Then the circuit was connected to the controller and the analogue input from the sensor is printed to the serial monitor. To test if the infrared sensors mounted underneath the robot were able to detect the black borders and cliffs, the sensors were pointed towards a piece of white paper. This paper had black tape to mark the borders and dark spots to mark cliffs (see figure 9). When the sensors were held over the white area, the signal received was high, while over the black tape, the signal was low.



Figure 9: Test model of search area

5.1.2 Range Measurement

To determine the range of the IR-sensors, we pointed the sensors towards some black and white paper. The distance between the sensor and the paper was increased until the sensor could not detect the difference in signal between black and white. As a result of the test and performance of the infrared sensors, it was concluded that the range the sensors is approximately 1.5cm.

5.2 Ultrasonic sensors

5.2.1 Object Detection/Tracking

Before the ultrasonic sensors could detect objects, the exact functioning of the sensor should be investigated. Based on this investigation some Arduino test code was written and the sensor was placed in a breadboard connected to the Arduino. First the Arduino sends a pulse signal to the sensor, which will transmit an ultrasonic pulse. The echo received by the sensor is directly send back over the same wire. The interrupt feature of the Arduino is used to measure the difference in time between sending and receiving the pulse. This time difference was then converted to a distance.

In order to make sure that the ultrasonic sensor mounted on top of the robot can detect an object, which is most likely a mountain since it is mounted on top. A test was conducted where the sensor was oriented to the front. An object was held in front of the sensor. The distance calculated by the Arduino is printed to the serial monitor. In this way the object detection could be tested. This test was successfully completed.

After this test a more advanced test was performed. The sensor had to sweep 180° in order to scan the area in front of it. If the robot detects an object it will then measure the distance between itself and the object. When it is done scanning the area and calculating distances it should drive towards the nearest object and stop in front of it.

In the actual test this did not work as well as planned. The robot had no problems detecting an object, but did not navigate as accurate as hoped for. It drove in the correct direction every time, but ended up exactly where it was supposed to be, just a few times. This was believed to be because the tables and floors used on this test are a bit slippery and hence the robot was sliding a little bit while driving on that surface.

5.2.2 Range Measurement

The test carried out to look at the range performance of the ultrasonic sensors mounted on top of the robot was the following: Objects were placed in front of the ultrasonic sensors. The sensors were again oriented to the front. An object was held in front of the sensor and the Arduino calculates the distance between the sensors and the object. Then the distance between the sensors and object was increased with a little distance until the sensors could not detect the object anymore. The maximum range was concluded to be approximately 3.5 meters.

5.2.3 Differentiation (Sample vs. Mountain)

To differentiate between a sample or a mountain the robot will use its two ultrasonic sensors. The principle is very simple. If only the ultrasonic sensor mounted at the bottom of the robot detects an object (at short distance), it is a sample, but if both sensors (top and bottom) detects an object, it is a mountain. A couple of tests were conducted to make sure that the robot could differentiate these objects.

First test was to check if both sensors, top and bottom, could detect a mountain. During separate tests both sensors worked, but when working together to detect a mountain, some troubles occurred. The robot was still able to detect a mountain using only the upper sensor. This sensor worked perfect, but in the first place the lower sensor did not work for some reason. After some investigating a short circuit was discovered and solved. Eventually both sensors worked and could distinguish samples from mountains.

The second test was to make the robot drive towards two objects, one mountain and one sample. The robot was able to detect the mountain by using the upper sensor and the sample.

5.3 Grabbing Samples

It was performed a three step test to make sure that the robot was able to pick up samples. The first step for the robot was to grab a samples laying right in front of it. This step was successful so the test proceeded to the second step which was to put a sample a little bit to the left of the

robot. The robot performed flawlessly in the second step as well. Third step was to put the sample a little bit to the right of the robot. It managed to pick up this sample as well without any problems, so it was concluded that the task of grabbing samples was working as planned. During this test the robot was only asked to pick up a sample.

5.4 Returning to the lab

In order to determine if the robots are capable of returning to the lab, two separate tests were carried out. The first one was to check if the robot could drive up the ramp and drop off the sample. Both robots had to go through the test, but only one at a time. The robot was put in front of the ramp (without a sample) and then "asked" to drive up the ramp. The robot succeeded in this task and was asked to do this again, but this time while holding a sample. This test was successful as well for both robots, so it was concluded that the two robots should be able to perform this part of the task without any problem.

The second test was to check if the robots were able to get back to the lab and in front of the ramp. This test was carried out in the following way: One robot was placed at a random position a couple of meters from the lab. Its job was then to locate the lab using the lab tracking system mentioned in part 3.5 *Research lab*. The infrared transistors mounted on top of the upper ultrasonic sensor is supposed to detect the infrared light emitted from the beacon at the lab. The robot managed to pick up the signal from the beacon and navigate itself in that direction. While black tape was used at the sides of the ramp in order to guide it towards the opening of the ramp.

5.5 Navigation

In this test a copy (like the one on figure 9) of the surface used in the final report was used. This was a two-part test where we first tested if one of the robots behaved as expected, which means that it moves in the correct directions and return to the lab with the rocks after both robots were tested separately, they were tested together. After conducting the tests in 5.4 *Returning to the lab* it was settled that the robot could locate and navigate towards the lab, but in this test it had to do that after navigating throughout the whole search area, grab a sample, and then manage to navigate back to the lab.

The two robots have their own algorithm on how to search the area, these algorithms are explained in 3.8 *Algorithm*. This means that they have different ways to navigate and therefore a test of both robots separately was important to conduct. These separate test runs were carried out several times and the robots did perform somehow as we wanted in some of the test, but in some of the test runs the robots failed to perform. As expected regarding the results from the test runs with only one robot in the search area, there were a few problems getting both robots to work together in the search area. At the end of the project period, one of the robots experienced problems with the servos and the Arduino and could not be used in the presentation. How that might have happened and possible solutions to solve the problem will be discussed in part 7. *Discussion*.

6 Conclusion

Based on the results of the tests and final demonstration, we can conclude that the proof of concept works. Our robot was able to sense its environment, and was able to detect mountains and cliffs using its sensors. It showed that it could avoid these obstacles while trying to recover its original path. Moreover, the sophisticated algorithms showed that the robot could (although not entirely accurate) orient itself and find samples in its environment. The one design requirement which was not entirely fulfilled was the returning of the research samples to the lab, as this proved to be a very difficult task with the sensors used. In order to make such a robot really suitable for a mission to Venus, the sensors and actuators need to be a lot more accurate, the algorithm has to be more flexible and the robot itself needs to be more robust. This will be covered in the discussion.

7 Discussion

7.1 General

First of all, the day before the demonstration the robot programmed with the Mountain-navigation algorithm broke. Tests revealed that this was a problem with the shield right above the Arduino unit. In isolation, the Arduino still could run simple programs. Once the robot was connected to power, the servos would start jittering and the sensors would return gibberish. The cause of this problem wasn't found due to the short time to the presentation.

During the project, there was also difficulty with the Arduino Uno USB port. It was only discovered in the last week that this port was broken and half of the uploads had failed. To upload a program, the USB-cable had to be firmly pushed upward into the USB port of the Arduino in order to receive Serial signals and upload programs to the ATmega chip. The rpm sensors of the wheels were upside down from the day they were obtained. This returned useless data. Checking each component sooner would have saved a lot of time debugging.

Finally, the group behind this project consists of 5 instead of the usual 6 members. It was soon discovered that the work load was too high in the timeframe the project had to be finished by. In a future iteration of this project, a larger team should be assembled.

7.2 Software

The algorithm hits some limitations of the Arduino Uno. Limitations found while developing this project were: amount of timers, amount of inputs, clock frequency, program size, amount of cores/threads. In a future version of this project, it would yield better results to use an Arduino Mega and/or a Raspberry Pi.

7.3 Hardware

The circuits for the infrared sensor were made on stripboards. This caused short circuits and unreliable connections at times, which led to problems examining the surface. It was by no means easily doable to make these circuits on a printed circuit board, but that would have been far more reliable.

8 Contributions

8.1 Final report

<i>Section</i>	<i>Written by (Mainly)</i>
Introduction	Loek
System-level description	Loek
Component-level description	General: Kurt, additions by Loek Lab: Marius Circuits: Tim & Niels
Strategy & algorithm	General: Loek, additions by Kurt Mountain-algorithm: Loek Sweep-algorithm: Tim
Testing	Marius
Conclusion	Marius & Loek
Discussion	Software: Kurt Electronics: Niels

8.2 General

<i>Task</i>	<i>Worked on by (mainly)</i>
Electronics (soldering circuits, acquiring components, making modifications to the robot in general)	Tim, Niels & Marius
Main algorithm	Kurt & Loek
Documentation (while the project was being developed, mainly about test results)	Marius