DES rover project

Project Proposal

Gerhard van der Knijff (s1006946), David Vonk (s4681533)

December 11, 2020

1 Introduction

This report makes a proposal for the requirements of our Mars Rover project for the Design of Embedded Systems course. We have given the requirements recursive labels "subject.specification.sub-specification". First we list our requirements, then the priorities of those requirements. Finally, we propose a brick-layout for the physical robot and explain why we would choose this specific specification.

2 Requirements

We want to design an expressive DSL that does not provide too much complexity. This means that the programmer is naturally limited in the things they can do, while those things that are possible are more easy. Nonetheless, this requirements list should allow for a fair amount of flexibility and the ability to perform most missions.

We defined the priority of the requirements using the *M*ust-, *S*hould-, *C*ould- and *W*ould like to- have format.

In the list below our requirements proposal can be found. Some of the defined priorities are noteable:

- The measurements are all prioritised as *S* or lower. This is because it does not add any useful functionality to the capabilities of the rovers besides reporting its status to the programmer. Since the robot cannot act on any historical measurements, it is enough to add them as conditional. Outputting measurements and other things (see also user.feedback)does have some debugging value to the programmer.
- An extension to the listed requirements could be to include statefulness (variables) and boolean algebra in the conditions. This is, however, likely too difficult given our time and possibly even the abilities of xtext.

user The rover should be programmable by a 15 year old person, who has little experience with programming.

user.endproduct The endproduct should contain at least two python files that the user can use to run the rover.

user.intuitive The controlling DSL language should be intuitive to use and closely follow natural language.

user.highLevel The DSL language should be high level, and not require the programmer to think about complex problems such as communication between bricks, controlling individual motors and sensors, collision avoidance and staying within the borders.

user.validation The user should be notified if the system detects undefined or erroneous instructions

user.independent The robot should perform the missions independently, without any human interaction

user.interupt The user should be able to stop the robot at any time.

user.feedback The robot should give the user notifications of its status.

---

safety The robot should move through the field to complete the missions, without being stopped by the environment.

safety.borders The robot should stay between the white borders.

safety.recovery The robot should be able to go back to a safe state if the borders are detected

safety.speed The robot should move at a speed no greater than the speed at which it is able to detect the border and stay within it.

---

missions The robot should be programmable with missions.

missions.examples The robot should be able to do the following three missions: 'Find and probe all lakes', 'probe behind the border on each side of the field' and 'do a random walk until an obstacle is found'.

missions.mission A mission should be a collection of instructions which achieves a specific goal

missions.expression There should be enough freedom in instructions to define missions utilising all sensors and motors.

missions.expression.movement.forward Ability to let the robot move forward

missions.expression.movement.backward Ability to let the robot move backward

missions.expression.movement.rotate Ability to let the robot rotate on the spot (left and right)

missions.expression.movement.turn Ability to let the robot make a turn (move forward with right or left tendency)

missions.expression.movement.wait Do nothing (most useful in combination with the time constraint)

missions.expression.conditions Ability to specify multiple conditions (distance, time, sensor value) which stop the movement if (un)met

missions.expression.conditions.distance Ability to specify distance condition for forward and backward movement

missions.expression.conditions.sensordistance.front Ability to specify distance condition for ultrasonic sensor on the front

missions.expression.conditions.sensordistance.back Ability to specify distance condition for ultrasonic sensor on the back

missions.expression.conditions.colorsor Ability to specify a list of colours that, if any are found, satisfy the condition

missions.expression.conditions.push.left Ability to register push on front-left push-sensor

missions.expression.conditions.push.right Ability to register push on front-right push-sensor

missions.expression.conditions.push.back Ability to register push on back push-sensor

missions.expression.conditions.time Ability to set a time countdown. After the set time, the condition is met.

missions.expression.measure.distance.front Make a distance measurement in front of the rover and output it to the user.

missions.expression.measure.distance.back Make a distance measurement at the back of the rover and output it to the user.

missions.expression.measure.distance.overBorder Make a distance measurement over the border of given color and output it to the user.

missions.expression.measure.color.left Make a color measurement on the left sensor and output it to the user.

missions.expression.measure.color.right Make a color measurement on the right sensor and output it to the user.

missions.expression.measure.color.center Make a color measurement on the center and output it to the user.

missions.expression.measure.push.left Measure if the front-left touch-sensor is pressed and output it to the user.

missions.expression.measure.push.right Measure if the front-right touch-sensor is pressed and output it to the user.

missions.expression.measure.push.back Measure if the back touch-sensor is pressed and output it to the user.

missions.expression.measure.probe Use the probe and output result to the user.

missions.expression.measure.probe.overBorder Move over the border (lake) specified by a specific color with the probe and output the result to the user.

missions.expression.statefulness Ability to remember conditions that were met in past movements and use these for current movement.

missions.expression.boolean-algebra Ability to use all boolean logic in the conditions

missions.expression.variables Ability to use variables to store and reason with sensor measurements

---

plan The user should be able to define multiple consecutive missions, which the robot will execute sequentially

Requirement	Priority
user	M
user.endproduct	M
user.intuitive	M
user.highLevel	S
user.validation	M
user.independent	M
user.interupt	M
user.feedback	S
safety	M
safety.borders	M
safety.recovery	S
safety.speed	M
missions	M
missions.mission	M
missions.examples	M
missions.expression	M
missions.expression.movement.forward	M
missions.expression.movement.backward	M
missions.expression.movement.rotate	M
missions.expression.movement.turn	C
missions.expression.movement.wait	S
missions.expression.conditions	M
missions.expression.conditions.distance	M
missions.expression.conditions.sensordistance.front	M
missions.expression.conditions.sensordistance.back	S
missions.expression.conditions.colorsor	S
missions.expression.conditions.push.left	M
missions.expression.conditions.push.right	M
missions.expression.conditions.push.back	S
missions.expression.conditions.time	C
missions.expression.measure.distance.front	S
missions.expression.measure.distance.back	S
missions.expression.measure.distance.overBorder	C
missions.expression.measure.color.left	S
missions.expression.measure.color.right	S
missions.expression.measure.color.center	S
missions.expression.measure.push.left	S
missions.expression.measure.push.right	S
missions.expression.measure.push.back	S
missions.expression.measure.probe	S
missions.expression.measure.probe.overBorder	C
missions.expression.statefulness	W
missions.expression.boolean-algebra	W
missions.expression.variables	W
plan	M

3 Mapping sensors to ports

We propose the following configuration of the sensors and actuators:

	EV3 brick 1	**EV3 brick 2**
Actuators	Left motor	
	Right motor	
	Measurement motor	
Sensors	Touch left	Color mid
	Touch right	Touch back
	Color left	Ultrasonic front
	Color right	Ultrasonic back

EV3 brick 1 will be our master brick. We put the motors of the wheels here, because it is important that the master handles the driving fast and correctly. The measurement motor is also placed here, because in this way the master can handle this motor fast, and the communication via Bluetooth is minimal. For the sensors, we want to keep the total latency as low as possible. The sensors that are used the most, will therefore be on the master brick. We chose to put the left and right color sensor, and the left and right touch sensor here. These sensors are the most important sensors to guarantee the safety of the robot and will thus be used the most often. We think we do not need the color mid sensor here, because we will be able to detect the borders or lakes soon enough with only the left and right color sensor. The sensors on brick 2 will be used a bit less than the sensors on brick one. The US sensors already take more time to measure, the latency of the communication does not matter much for those sensors. If we need a distance check, we will send a request via Bluetooth to the other brick to get the current distance.

The configuration that will be used is the following:

	EV3 brick 1	**EV3 brick 2**
Actuators	Left motor	
	Right motor	
	Measurement motor	
Sensors	Color left	Touch left
	Color mid	Touch right
	Color right	Touch back
	Ultrasonic back	Ultrasonic front

We don't think this is the most optimal solution, because we wanted to have the ultrasonic sensor on the slave brick. We also thought that it was not necessary to have the color mid sensor on the first brick, because the two other color sensors would be enough to navigate.
We think that if the robot has a reasonable speed, the communication between the bricks will be fast enough to share the sensor values in time.

4 Work plan

We decided to split up the work. One of our team will be working on the DSL, and the other one will start coding the python files. For the DSL, the following requirements have priority:

1. The ability to create missions (user.missions)
2. The missions should contain instructions (missions.mission)
3. The ability to specify certain movements (forward, backward, etc.)
4. The ability to specify conditions (color, distance, etc.)

For all these requirements, we should keep the user in mind. The DSL must be intuitive and close to natural language. After adding these first requirements, the next things that will be added are the measurements. Validation can also be added.

For the python code, the following requirements have priority:

1. Communication between slave and master over bluetooth
2. Sensor readings by the master and slave (and communicating them over bluetooth)
3. Make functions for movements and rotations
4. Make functions that implement the more complex behaviours as combinations of basic movements and rotations as defined in the DSL.

5 Timeline

In the following timeline, we specify which requirements will be (partially) done in which week. In the progress part, we will describe if the requirements have been implemented in time.

Week	Requirements
Week 1 (18-24 november)	-
Week 2 (25 november - 1 december)	user, user.endproduct, user.intuitive, user.highLevel, user.validation, missions, missions.mission, missions.expression, plan
Week 3 (2 - 8 december)	safety, safety.borders, safety.recovery, safety.speed, forward, backward, rotate, turn, wait
Week 4 (9 - 15 december)	missions.examples, conditions, distance sensordistance.front and back, colorsor, push.left, push.right, push.back measure.probe, probeOverBorder
Week 5 (16 - 22 december)	user.independent, user.interrupt, user.feedback, conditions.time, measure.front, back and overBorder, measure.color, measure.push, statefulness, boolean-algebra, variables

6 Progress

In this section, we write down what we have done per week. We will denote if we have finished the requirements that were in the timeline, and if we are on track.

6.1 Week 1 (18-24 november)

In this week, we started writing the report. We created the requirement list and added our proposal for the mapping from the sensors to the ports.

6.2 Week 2 (25 november-1 december)

We updated the requirements a bit, and added the mapping from sensors to ports that was decided in the lecture. After that we split up, and worked both on our own part.
David started with the generation of the Python code. The first priority was to connect the two bricks, and create a master and a slave file. Furthermore, he created a utils file to get the data easily from the sensors, and which handles transmission of sensor values from slave to master. Gerhard started with the DSL. He generated a first grammar, and some instances of the grammar. That worked out well. After that, he also added validation to the grammar.

6.3 Week 3 (2 december-8 december)

In this week, we worked again on separate parts. Gerhard word on the code generator. This was quite a lot of work, to get the grammar into nice code. In the end, the code generator was almost done, and the DSL is transformed into code nicely
David worked on the code itself. This week were mainly about specifying functions that the generator can call. The idea was to make a clear documentation which Gerhard could use in the following week to refine the code generation and error prevention measures. Furthermore, David worked on the interpretation and execution of the configuration generated by the DSL.

6.4 Week 4 (9-15 december)

David worked on the code again. This week the goal was to implement basic movements and to debug the mission-execution system.

Given the way in which we handle the conditions within movements, the subsumbtion architecture is superfluous. The system starts the motors in non-blocking mode, and stops them when the conditions are met (or if the specified distance was traveled). A second thread which does condition-checking is thus not necessary: therefore this was removed in this version.

After experimenting a little bit with updating sensor values from the secondary brick, we found that simply requesting a new update as soon as the previous one is received gives very good results. This avoids congestion on the bluetooth connection, while still having a pretty good update interval. The frequency is also not too high: the sensorupdates do not interfere (much) with doing movements.

Although the sensor-update system works pretty well, we noticed that reducing the movement speed during more complex movements (e.g. near a border), where safety-sensor readings are important, removes most risks of driving off the map.

It may be worth it to do more experimenting with the slave-synchronisation frequency when the more complex moves are implemented, to see whether reducing this frequency improves border detection performance.

A lot of work was done on refining the details of the system: which things are handled where, and how conditions and sensors are specified. Colors for a colorCondition are, for example, now specified using a set instead of a list. This makes it easier to specify a set with all allowed options and check whether the desired colorset is a subset of the allowed ones. This is done the same for the targetsensor lists (so only where multiple sensors can be specified).

The current rover is able to move forward and backward (also in a safe manner), and functions with all of the conditions and measurements. It is therefore now in a very basic working prototype state. Now we can alternatingly add functionality and test. It is now also relatively easy to work on the code in parallel, using separate branches, where each of us works on their own independent function.