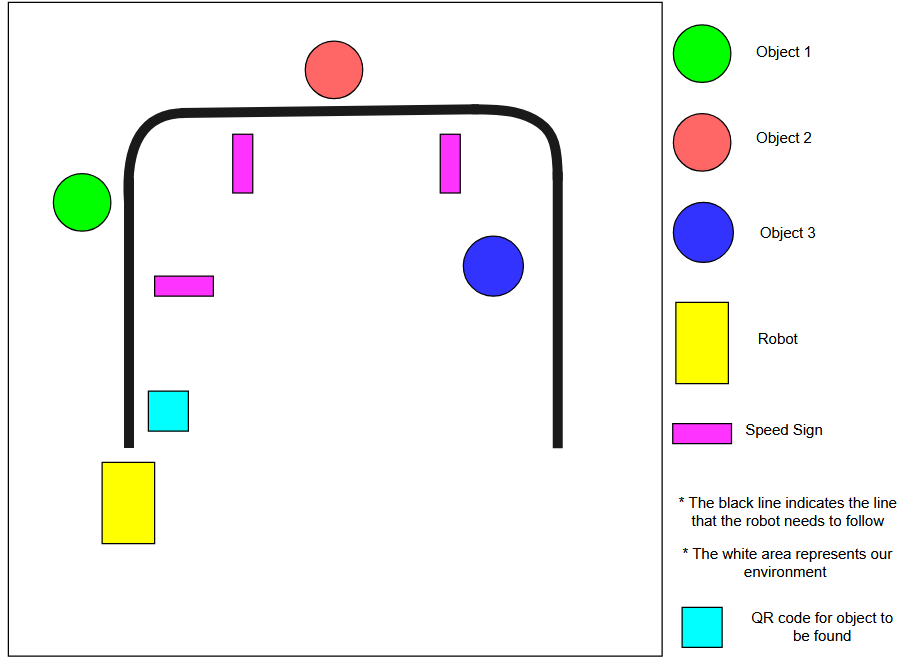
**Embedded Visual Control**

**Function proposal, deployment and validation – 50%**

1. **High-level description of the proposed function (proposal phase)**

* Textual description of the function to be developed;
* Use figures for better explanation and understanding;

Project description: We would like to program a robot which is able to find a specific object along a premade course. To determine which object the robot has to find, it first needs to read a QR code that specifies the object. It should be able to follow the pre-made course by following a white line and detect the specified object using object detection. Using OCR, we would want the robot to be able to change its speed at different intervals along the course. Finally, to see if the robot has reached the correct object, it should stop only at the correct object which it read from the QR code fully autonomously. We will use a poster containing black lines as our environment to set up our course, see figure below as a representation of a possible course.



1. **Requirements (proposal phase)**

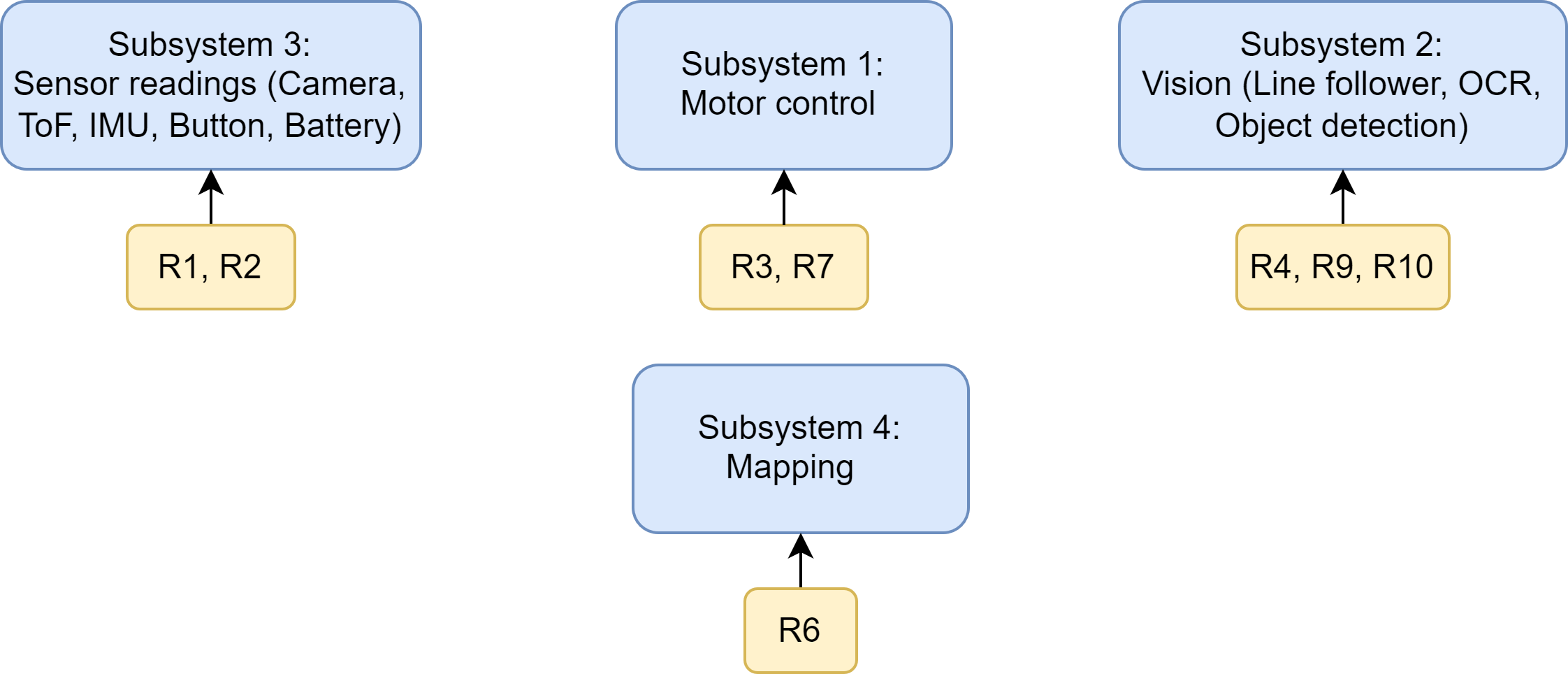
Based on the high-level description of the system, extract the system requirements to be met by the developed function; refer to the system description and figures to clearly specify the requirements.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Description | Success criteria | Priority (mandatory/optional) | Remark (task division) |
| R1 | Collision avoidance node: this node is responsible to stop the robot from moving when it is about to bump into an object. | Does not hit an object, it stops and reverses a bit to avoid the object. | **Mandatory** | (Martijn) |
| R2 | Closed loop control for correcting the trajectory of the robot whilst driving straight | Deviation from the straight path (error). | **Optional** | (Toan) |
| R3 | Integrate acceleration motion profiles. | Reduce wheel slip improve odometry and smoother movement. | **Optional** | (Martijn) |
| R4 | Make a line detection node that is able to detect a (white) line on a (dark) surface and generate move commands | It should be able to follow the pre-made course in the picture above. | **Mandatory** | (Steve) |
| R5 | The object needs to be found within 4 minutes | On the used track the object is found within 4 minutes | **Optional** | (general) |
| R6 | Map the course based on previous ‘runs’ and use this data to optimize motor speed. | A usable map of the course is created in software. | **Optional** | (Karsten) |
| R7 | Extend/redesign the motor control API with the correct functions so the robot is able to move in the environment in a robust way. | It moves to its target location without more than 2 restarts. | **Mandatory** | (Martijn, Luka) |
| R8 | When the robot starts following the line it needs to stay in its field of view till it finds the object | The line stays in the camera feed until the object is found | **Optional** | (Toan) |
| R9 | The robot needs to change the speed to the speed limit it reads using OCR. | When the robot drives by a sign it needs to change its speed | **Mandatory** | (Toan) |
| R10 | The robot needs to be able to detect blue, green, and red objects | When it drives by an object it needs to detect it and perform some action. | **Mandatory** | (Martijn, Luka) |

1. **Proposed approach (proposal phase)**

* Explain the method from theoretical viewpoint;
* Divide the proposed approach into functional sub-systems
* Map the sub-systems to the requirements and responsible group members

The whole robotic system can be divided into 4 high-level subsystems. The first one being the subsystem that controls the motors. It exposes an API in the form of different ROS services which can be called by any ROS node in the network to control the motors and thus the movement of the robot. It also has the Odometry node which reads the encoders for positional feedback which some functions use e.g. if a node calls to motor control function “move one meter” the system should know when it travelled one meter. The second subsystem is the vision one; this system oversees processing the camera feed and determining what actions need to be taken. The line follower node will be the main decision maker node as it uses the motor control API to move the robot based on where the line is. The OCR node will only send max speed changes to



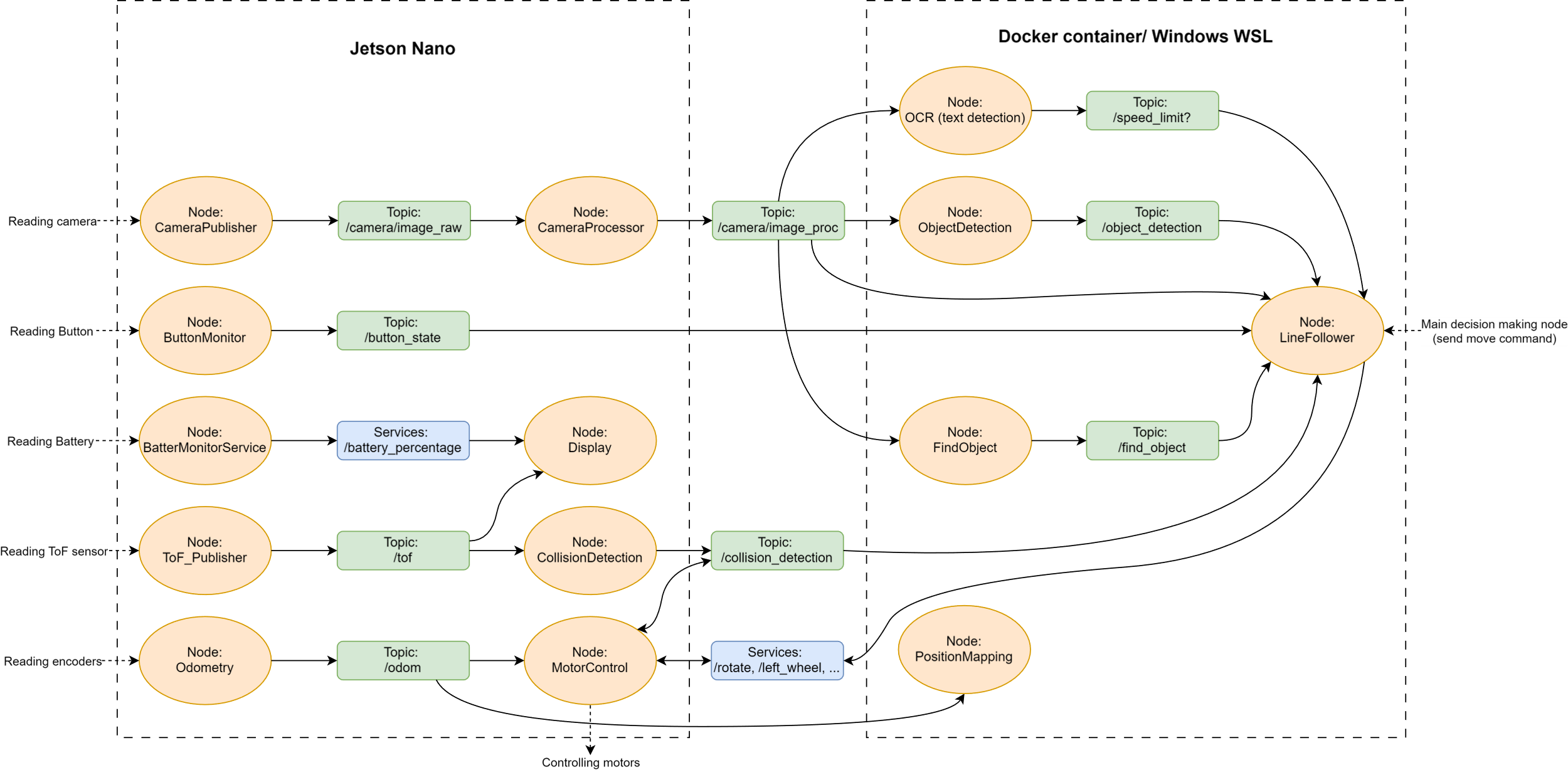
1. **Sub-system implementation and validation (final submission)**

* Details of the deployment of each sub-system
* Explain the functional architecture with details of ROS nodes and topics
* Report possible performance numbers associated with the requirements, e.g., timing, throughput, FPS, accuracy, distance etc.

The deployment of the subsystems can be split into separate parts:

* Subsystem 1 is run on the jetson robot due to it being the main motor control node
* Subsystem 2 is run from separate devices through a docker container due to their computational requirements
* Subsystem 3 is run on the jetson robot to have access to all the sensor readings required for the robot to operate
* Subsystem 4 is run from a separate device.

The functional architecture can be split into multiple separate nodes, topics, and services. Services are an alternative method for sending and receiving data from separate nodes. However, the services are predominantly used for sending motor commands to the motor control directly. Each service has a separate function, for example there is a service which sends a message to the motor control to continuously rotate in one direction or the other, or to continuously move in a straight line. These services were used in the main motor controller, which was the line follower (as the line follower acted as the control hub of the robot). Alongside this, a set of topics were used to connect smaller nodes to larger nodes which enabled them to have more specific behaviour and send this message when a certain scenario is detected. For example, in the case of the OCR, if it detected a number that signifies a change in speed, it would send a topic with said number so that it could be relayed to the main controlling algorithm for it to use. For the sake of convenience and ease of coding, most of the functions with different properties were separated into different nodes. For example, the line follower and object detection were two distinct nodes. A clearer representation of the nodes, topics and services can be found here:



Performance metrics:

The video FPS was limited to 15 to minimise the chance of the camera crashing.

The number of motor commands sent from the line follower was 20 commands per second.

Accuracy: There is no metric of accuracy that we present, however the closest thing would be the error calculation used in the line-follower. However, this is just used to indicate if the line has turned

Distance: The testing environment used to show the robot functionality was roughly 2 meters in length

Timing: The robot is able to complete the testing environment in around 20 to 30 seconds.

1. **Integration and verification (final submission)**

* Describe the integrated deployment of the sub-systems
* Refer to the requirement table and fill in the following table;

|  |  |  |
| --- | --- | --- |
| ID | Satisfied | Remark with specific details |
| R1 | **Yes** | The written software works well, but the sensor doesn’t give the best data. (specially when the sensor doesn’t see anything in its range is produces junk data that is also less than 8000 e.g. 200 which makes collision detection only work when it almost collide with something. |
| R2 | **Half** | The robot drives straight using a PID controller, a correction system using the IMU or camera is lacking. |
| R3 | **Yes** | Works fine, you can specify acceleration and deceleration values as parameters. |
| R4 | **Yes** | Line follower is able to follow the line however it can not do more complex line following actions (such as very tight turns or multiple lines that cross each other). Furthermore it does not do well on |
| R5 | **Yes** | On the final video it can be seen it takes less than 4 minutes |
| R6 | **Half** | A map of the course can be created, however the speed is not optimized. This second half was removed in favor of showcasing the OCR reading and changing speed. |
| R7 | **Yes** | The motor control now uses services instead of topics for it control, the services provide extra feedback so the sender knows if the command is accepted or not. |
| R8 | **Half** | Technically this specific function was not implemented, however it semi included due through the line follower |
| R9 | **Yes** | If it detects the words “speed” on a card followed by a colon and after it sees a value the speed will be changed to the detected value. |
| R10 | **Yes** | It detects objects based on color, it works well in environments which do not feature the color it wants to detect. As you may expect it doesn’t work well when its in an environment that features the color it wants to detect as it then also detects other things as “object”. |

1. **Link to the code and demo video (final submission)**

* Give access to the public code repository and the demo video

Link to the Public code repository: <https://github.com/Martijn-Strolenberg/evc_group_4>

The Final code for the project is found in the folder **final\_project,** but also as a part of this submission

To run the code the following steps should be done:

On the Jetson:

After sourcing the project\_group4 folder inside the final\_project folder. The following commands should be run in the terminal:

**roslaunch motor\_control driving.launch**

**roslaunch sensor\_readings display.launch**

**roslaunch camera camera\_undistorted.launch**

On the docker (or whichever medium is used)

Also after sourcing the project\_group4 folder run the following commands:

**python src/camera/src/line\_follower.py** (without OCR) or **python src/camera/src/line\_follower\_OCR.py**

**python src/camera/src/camera\_qr.py**

**python src/camera/src/object\_tracking.py**

The video submissions can be found in the **VideoSubmissions** folder where the **FinalVideoSubmission** folder contains integrated line follower, qr scanner and object detection, while the **IndividualVideoSubmission** folder contains separate videos of the individual parts.