Student Name: Boyang Xia **UoB Student Number**: 1902964

1. **Briefly describe the architecture of your system for the line-following task. Identify which elements of the system have the greatest influence on the level of performance of your Romi to complete the line following task.**

1.1. Overview of Task & System Architecture

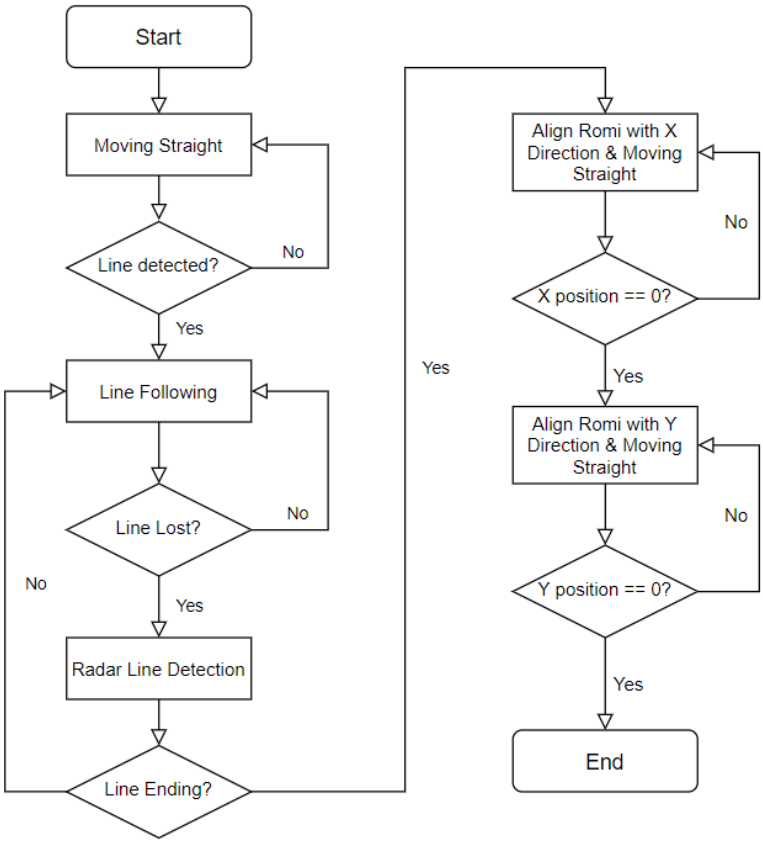

Figure 1. Flow Chart of **Line-Following (Left)** & **Going-Home (Right)** Tasks

The Romi **starts with calibrating the sensors**, since the value may vary from different external conditions, such as light, the texture of the map, and cleanliness of the map surface; and **calculating kinematics**. After calibration, Romi moves straight then switch to the line-following pattern when the line is detected. Line-following pattern is accomplished through **Bang-Bang control**. If Romi loses the line, it would switch to **Radar Detection** mode, trying to re-join and follow the line again. If Romi actually arrives the end, then it would rotate and align itself with the X-axis, then move backwards and straight. If Romi reaches the position where X = 0, then rotate to align it with the Y-axis, then move backwards and straight again till Y = 0, then **stop**.

System Performance: Romi **successfully returns home** and stops within the **blue circle**. Here is the link of Video Record.

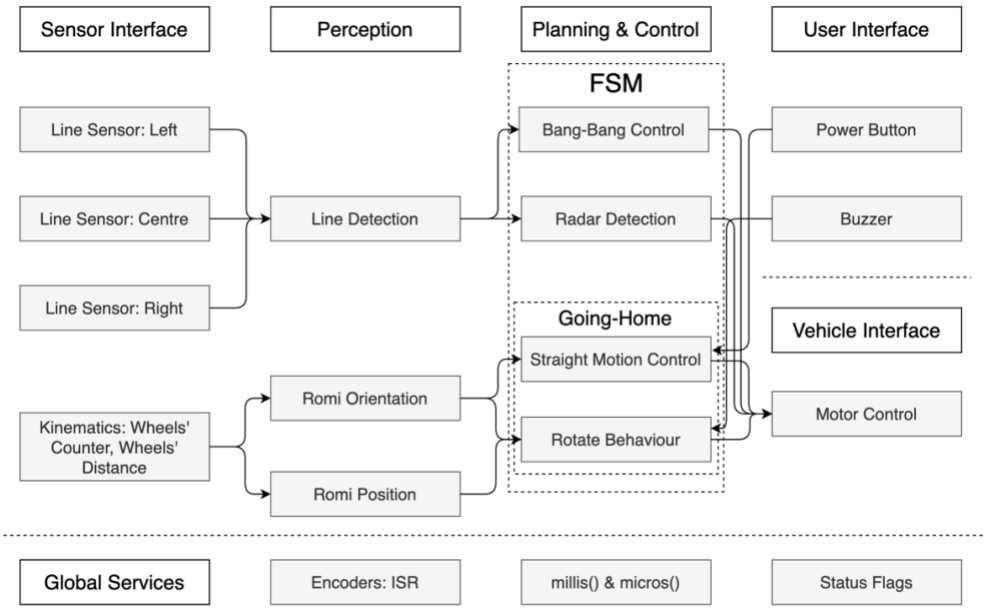

Figure 2. System Architecture of Romi

1.2. Key functions of the system and details

[1]. Kinematics;

[2]. Straight Motion Control;

[3]. Bang-Bang Control;

[4]. Radar Detection Behaviour;

[5]. Rotating Behaviour.

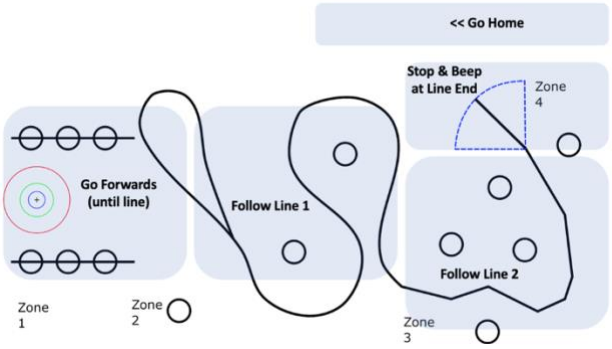

Figure 2. Default Map

As Figure 2 illustrates, many functions of the system are not overlapping to each other, especially when the PID control strategy is not used. Decomposing the task into parts as listed before, sub-tasks are combined using FSM, would **reduce the complexity of parameter tuning, testing and debugging**.

Kinematics covers the whole task since Romi is started, which calculates the position and orientation of Romi, as the basis of returning home. **Straight Motion Control** is activated on other sections except for the line-following, such as Line Searching and Going-Home, more details are demonstrated in Q2. **Bang-Bang Control** dominates both the line-following parts: the curve line and polyline. Line following section relies on the line sensors, in which case sensor calibration is necessary. When Romi "believes" the line is lost, it would rotate itself at the last position, anticlock- and clock- -wise to detect the line, which is called **Radar Detection Behaviour**. **Rotating Behaviour** is used before and during returning, which aligns the Romi with coordinate axes.

In this system, **PID control is replaced by Straight Motion Control**, which is more intuitive and proved to be competent to finish the line-following task and return home accurately. **Radar**

Detection Behaviour is added later since I found that Romi sometimes loses the line, especially in the polyline part, this function could increase the robustness on passing the polyline. Rotating Behaviour is one of the essential parts of Going-Home in the right-angle path, which helps obtain accurate performance since it doesn't need to calculate the angle at the end of the line.

The most successful element of the system is Straight Motion Control. In this system, the Straight Motion Control is accomplished via controlling motors using the angle of Romi's orientation. Before moving straight, the system records the initial angle, which subtracts the current angle value to detect whether Romi is off the track or not. If the subtraction is positive, then drive left; otherwise, drive right. Straight Motion Control based on orientation is logically easy to understand and timesaving on parameter tuning. Except for the binary logic, the only parameter needed to tune is the power change of each motor, which cause the speed difference and change the orientation. More details are explained in Q2. This element is also the foundation of successfully returning home.

The element received the most development time is Going Home section, which is the most challenging part of the whole task. Because it relies on the accuracy of Kinematics, ways of rotating and whether Romi can perfectly drive straight. More details are explained in Q2.

The most valuable experience of the system design is decomposing the system and combining each sub-task using Finite State Machine. The reason I strongly recommend is it could help designers logically comprehend the function of each part and the relations among them.

2. Describe in detail a specific challenge or success relating to a sensor, motor control or a sub-system (e.g. behaviour) for your Romi robot.

Going Home is one of the big challenges, solved by being decomposed into two parts. The two problems are how to rotate accurately, and how to drive straight perfectly.

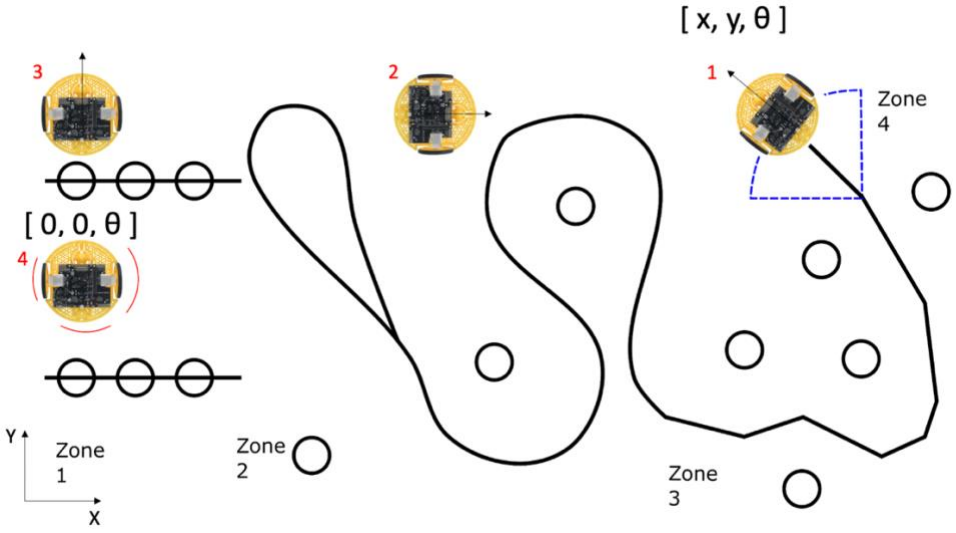

Figure 3. Going Home Demo. *Step 1*: Stop and Adjust the orientation; *Step 2*: Moving straight along with X-axis; *Step 3*: Stop and adjust the orientation; *Step 4*: Moving straight along the Y-axis and stop.

2.1. Rotate using orientation angle

```
void fir_rot()
{
  float speed = 20.0f;
  initial_theta = pose.get_theta();
  if ( pose.get_ypos() < 0 ) // default version of map
  {
    if (initial_theta < 0)
    {
      leftMotor(speed);
      rightMotor(-speed);
    }
    else
    {
      leftMotor(0.0f);
      rightMotor(0.0f);
      state = 4;
    }
  }
```

Going Home is accomplished through two paths: first **returning along X-axis**, then **Y-axis**. Therefore, the first challenge is how to align Romi's body with the axes, i.e. X-axis, before driving straight. As shown in the left, whose function is turning Romi, at the end of the line, to align its head-tail axis along with the X-axis.

Before rotating, initialise the theta as the current orientation value. In the default version of the map (the one showed in the Video), when Romi arrives the end of the line, it would **locate at the negative part of Y axis**, and **the value of orientation angle is negative**, then Romi rotates clockwise to turn its tail towards the negative X-axis, stop when the angle value is negative. After Romi drives straight and reaches the Y-axis, it would rotate anticlockwise to turn its tail towards the starting circles. By the way, modules `leftMotor` and `rightMotor` are pre-defined, whose inputs are value of power which drive motors. Motors rotate towards when the input value is positive; otherwise backwards.

2.2. Driving Straight

```
float theta_delta = initial_theta - pose.get_theta();
int speed = 30;

// theta_control starts:
int PWM = 0;

if (theta_delta > 0)
  PWM = -2;
else if (theta_delta < 0)
  PWM = 2;
else PWM = 0;

int left_demand = -speed - PWM;
int right_demand = -speed + PWM;

leftMotor(left_demand);
rightMotor(right_demand);
```

After aligning the Romi with the X-axis, it would drive straight towards the starting position side. Before driving, the system records the initial value of orientation angle, as `initial_theta`. In the process of driving straight, Romi calculates the change of orientation angle by subtracting the current value of angle `pose.get_theta()` with `initial_theta`, as `theta_delta`, where `pose` is the call of function Kinematics. If it is positive, then Romi rotates anticlockwise, otherwise rotates clockwise. This function relies on the accuracy of Kinematics. To look deeper, this behaviour is much more similar to Bang-Bang Control used in the line-following section. In stage 2 (Figure 3), when Romi reaches the Y-axis, or the absolute value of `y_pos` is within a range, then Romi stops; In stage 4 (Figure 3), when Romi reaches the X-axis, or the absolute value of `x_pos` is within a range, then Romi stops.

2.3. Advantages, disadvantages and reasons of choice

First, decomposing the Going-Home section in such ways could improve the accuracy of returning home. Because there is no need to calculate the angle between the end of the line and the starting point, therefore, it simply relies on the accuracy of Kinematics.

Second, using Straight Motion Control by controlling orientation angle rather than PID control, which is intuitive, computationally effective and avoid the time-consuming PID parameter tuning.

While, the major disadvantage of using this approach may cause the Romi drives in jerky behaviour when the power change, `PWM` is not tuned well.

Considering the above factors, the way of returning home is decided as in the design.

2.4. Other possible alternatives

Before this solution, driving Romi back home directly is tried, and in terms of controlling Romi's rotation, the wheel's step counting was contemplated as well. PID control was considered but which didn't go further. Content of lectures indicates that the counts of each wheel can be used when Romi rotating, which requires the high-accuracy measurement of wheel's diameter, and parameter tuning about the motors' power is time-consuming as well. Therefore, it is not used in the final version.

3. Describe a challenge in robotics you would like to further investigate. Relate your ideas to your experience of working with the Romi for the line-following task.

The problem I would like to explore related to the current task is autonomous cars with HQ cameras on the roof. The hardware would be switched to a real-size car with HQ cameras and distance sensors fixed on the board. The software would be used is not clearly identified but whose functions are more similar to those of NVIDIA DRIVE Platform. The environment is a broad square with some dark lines attached on the ground, a flag is plugged on the starting point, and some obstruction objects placed on the line path. Figure 4 illustrates the environment and hardware in brief, in which the icon of real-size car is replaced with the Romi. The task in overview is extended based on the line-following task: path following and obstruction avoidance using the computer vision.

The details of the task requirements:

- Start;
- Cars moves and searches the line to join, the driving pattern is not specified;
- Objects are placed on the path, such as humanoid toys and plastic-made blocks (see Figure 4). Therefore, autonomous car should behave like real cars with drivers: stop and wait when confronting humanoid toys, slow down and avoid when confronting blocks;

- A yellow box is placed at the end of the line, which is the target that car is required to fetch. Car needs to recognise it and stop for at least 5 seconds (imagine the box is loaded on the car by others), then drive home;
- There is a flag on the starting site, when car starts to return home, make it drive straightforward to the flag;
- Stop the car at the starting site;
- End.

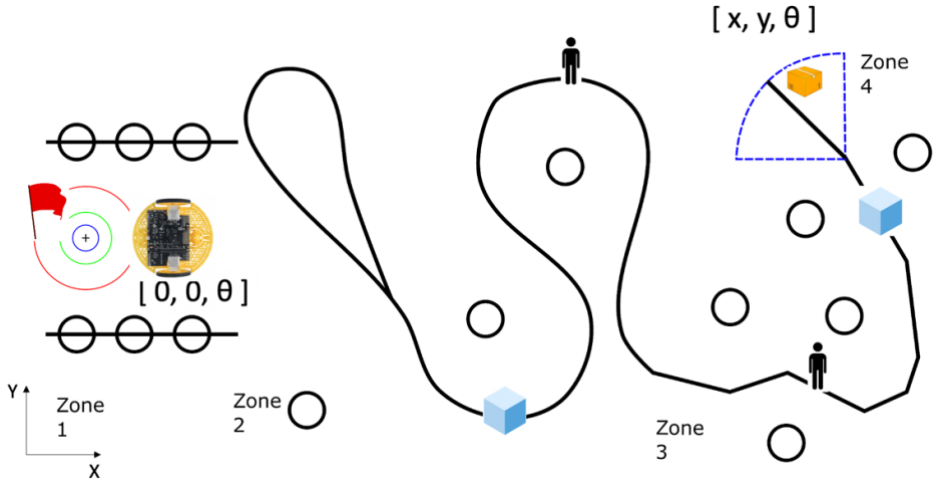

Figure 4. Environment of autonomous car research

Possible Solutions:

- Using computer vision for detecting and recognising objects, such as flag, block obstacles, humanoid toys and the targeting box;
- Using distance sensors attached on the car to detect the distance between the car and possible nearby obstructions, in order to avoid collisions;
- Motor control strategy based on Neural Network approach, which drives the car.

Performance Measurements:

- Whether the car is driving along the fixed path;
- Whether the car is capable of detecting obstruction objects. If detected, then beep;
- Whether the car is capable of recognising humanoid toys and plastic blocks. If detected, check its behaviour;
- Whether the car is capable of detecting the target box. If detected, then stop;
- Whether the car is capable of returning to the starting point and how accurate is the return.

One of the possible challenges of accomplish this task is how to combine Neural Network with Motor Control based on computer vision. The most valuable experiences of working with Romi line-following task is Finite State Machine.