Student Name: Jinhang Zhu **UoB Student Number**: 1950951

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.**

(a) *Provide a flow chart, diagram or pseudo-code to help describe your robotic system in overview.* Check the algorithm 1.

(b) *Describe why you decomposed the line-followinging task into your solution the way you did.*

The *reason* why I decompose the task into the solutions is that I desire a robotic system with high stability and low jitter in motion. The *objective* of the design is to **eliminate redundant motion to reduce the tracking errors** under the precondition that the Romi car can finish all the jobs.

Romi car is required to start with the calibration of the line sensors and cross the Zone 1. It is not known how long the car would take to join the line due to various speed configurations. Thus, it needs to finish the calibration before it starts unless it might perform the calibration task on the black line. In order not to mistakenly join the loop in Zone 2, the car should take linear motion in Zone 1. From the experiments, I observed that the car swayed to the other side before it resumed the right orientation to follow the line. The swings lead to redundant errors, and changes should be made to avoid this. Zone 2 and 3 have different types of lines: the smooth-shape road and the zigzag road. The car has to make a trade-off between running fast with lower finish time and slowly passing the zigzags without derailment. The following task is to leave the line over the endpoint stably. The car needs to pause for a moment while reminding us of the finish of line-following task with a beep sound. Then, I decompose the going home job into two parts: rotation and linear motion. The car has to make sure it faces the starting point unless it keeps rotation. In this case, the car may perform redundant rotation, so changes are made to assure a rotation angle less than 180°. For my solution, the car can go straight with minor errors, so the one-shot linear motion to return home is allowed. Finally, the car terminates all tasks and presents a static pose even if the loop is running in its microcontroller.

(c) *Which element of your system was the most successful, and why?*

I give credit to *Joining the line* task. Reasons are given as follows.

The way I implement this is to make sure the car starts line following at the moment it is about to run off the trail. When the car was running in line-following mode, experiments showed that it might join the line with some intense jitters over the point of contact. The jitters happen even when the speed is set low since the PID controller would magnify the errors and result in a considerable difference in the velocities of two wheels. To solve this problem, I applied **two contact detections** to enable the car to move onto the line smoothly without swaying in the wrong direction. The car runs in a straight line until it feels the trajectory **with the different side**. My solution in programming is to memorise the heading error when the car feels the line and iteratively observe the product of the current heading error and the saved error. It is time that the car should follow the line when the product is not positive.

My solution successfully eliminates the jitter motion and reduces the potential accumulation of tracking errors. Few oscillations take place in the motors so that the car may measure the velocities with higher confidence. The only extra memory my solution needs is a variable to store the heading error at the first contact with the line. After all, the Romi car is running in a simple environment, so a simple but effective solution is preferred.

(d) *Which element received the most development time, and why?*

I spent the most development time configuring the *line-following* part. The general problem of configuring the line-following part lies in **making the part cooperating with other elements of the robotic system**.

Algorithm 1 Workflow of Romi Car

```
Initialise $stage := 0$, $v$, thresholds, PID controllers, setup and calibration.
loop
    $S \leftarrow$ line sensor values, $m = \frac{S[0]-S[2]}{\sum_i^3 S[i]}$ (update the heading error)
    Switch($stage$):
        Case 0:                                          ▷ Go straight and feel the line
            repeat
                Go straight $v_{left} = v_{right} = v$ with $pid_{\text{straight}}$.
            until $\sum_i^3 S[i] \geq T_{online}$
            $stage \leftarrow 1$, $e \leftarrow m$ ($e$ note the side that feels the line)
        Break
        Case 1:                                          ▷ Joining the line
            repeat
                Go straight at lower speed $v_{left} = v_{right} = v - \Delta v$
            until $m * e \leq 0$ (when the other side feels the line)
            $stage \leftarrow 2$
        Break
        Case 2:                                          ▷ Line following
            repeat
                $pid_{\text{linefollow}}$ controls the heading.
            until $S[i] < T_{offline}[i]$, $\forall i = 0, 1, 2$
            $stage \leftarrow 3$
        Break
        Case 3:                                          ▷ Stop and beep
            Beep, $stage \leftarrow 4$, $\theta_{pos} = \arctan 2(-Y, -X) * 180/PI$
        Break
        Case 4:                                          ▷ Turn to face home
            repeat
                Rotation
            until $|\theta_{Current} - \theta_{pos}| < T_\theta$
            $stage \leftarrow 5$, $d_{dest} := \sqrt{X^2 + Y^2}$ (current distance from home)
        Break
        Case 5:                                          ▷ Go towards home
            repeat
                $v_{left} = v_{right} = v$
            until $d_{pass} > d_{dest}$
            $stage \leftarrow 6$
        Break
        Case 6:                                          ▷ Returned home, stop
            Beep, $v_{left} = v_{right} = 0$
        Break
    EndSwitch
    Speed control with $pid_{\text{speed}}$. $kinematics$ update.
```

The objective reality is the **high proportion of the distance of the line-following part**. The duration of one experiment of the line-following trajectory is long and irreducible.

Designing the system architecture is another tricky problem. Initially, the assumption was made that IF-Else better suits the semantic characteristics: if the car has finished the join-line part and has not ended the leave-line part, it then is performing line-following part. However, when it comes to robot programming, If-Else architectures introduced many flag variables and an increase in the use of memory. Another problem of If-Else is the increasing difficulty in debugging the codes. Switch-Case architecture only needs one variable *stage* and the codes are simple. Experiments took fewer efforts, given that the program became clear and well-formatted.

In terms of **the algorithms that determine if the car is online**, I initially chose the advanced Bayes Filter to compute the confidence of being online iteratively. However, the BF algorithm took a large memory to store the hyperparameters of the probability distributions (four 4×4 matrices) such as the probabilities that the car is at the new state after it takes action at the previous state. The main problem of BF in my case is the slow reaction of mode change: the car still goes straight after joining the line except running at low speed. Then I switched to the simple thresholding methods, and it turned out to be helpful.

Tuning PID parameters has always been a time-consuming task, especially in the case where there are four cooperated PID controllers. They work in a subsumption architecture. Two PID controllers for heading come into effect at different stages while the other two are at a higher level. I took the most of the time configuring two jobs of tuning. One is to choose the correct time intervals for controllers. The other one is to tune K_P, K_I, K_D. While tuning heading PID, I assumed that K_P should be large enough to pass the zigzags so that I can set high speed to reduce running time. However, the car can experience intense oscillations on the smooth road. To solve the problem, I lower down K_P and applied K_I to timely compensate the static errors. It turned out that without configuring K_D, i.e. $K_D = 0$ for all PID controllers, the car can operate in a smooth motion with low localisation errors.

(e) *What working practices would you recommend to others engineering a robotic solution for the same task, and why?*

I would have two suggestions. The first one is to **take a preview of the system architecture**. For assessment 1, I implemented a system which performs only the line-following mode. It took time to transfer the program from a prototype to a system. Therefore, I would advise others to prepare abundant knowledge of how the system operates and how each element cooperates before they started coding. The second suggestion is to **take an insight into the data while debugging**. For beginners, it is always hard to understand where the problem takes place. They are not encouraged to figure it out just by staring at the codes. In my case, many bugs in PID code snippets are observable with the help of sensors. One should perform experiments to watch the values of the variables, the sequence of the workflow.

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.** *You are encouraged to use small extracts of your source code where it is relevant, e.g. to communicate a problem, your solution, or how you captured debugging information.*

(a) *Choose an element of technology (hardware or algorithms) which was interesting or challenging for you. What were its advantages and disadvantages?*

One element of the solution is the **PID cascade control** for the *straight-line* motion. The structure is demonstrated in the flowchart 1. The output of the outer PID serves as one (demand) of the inputs of the inner PID. The demand for the outer PID is the zero orientation (i.e. the angle between the line of sight and the x-axis). The orientation is measured via the kinematics update. Meanwhile, the state of the movement is also measured. The kinematics or

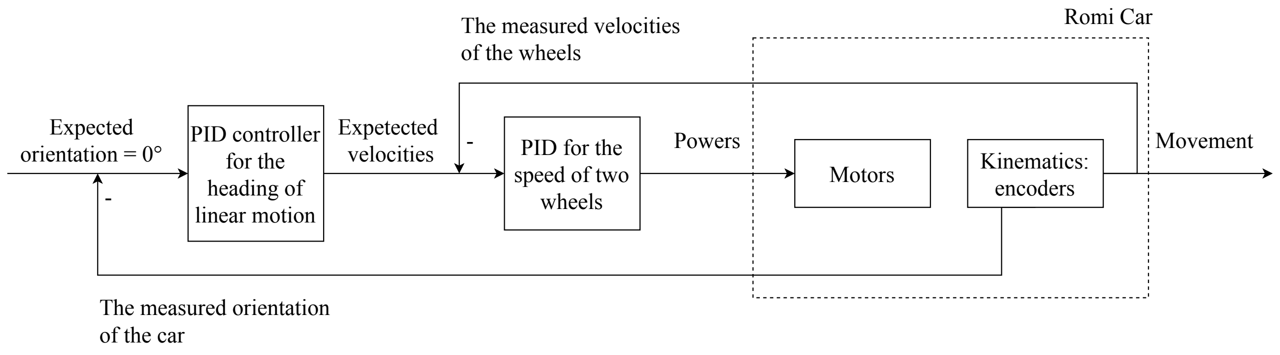
The measured velocities of the wheels
Romi Car
Expected orientation = 0°
PID controller for the heading of linear motion
Expetected velocities
-
PID for the speed of two wheels
Powers
Motors
Kinematics: encoders
Movement
-
The measured orientation of the car

Figure 1: Cascade PID Control of Linear Motion

encoders provide the tracked positions (X, Y) and the velocities of two wheels v_l, v_r. Notably, this algorithm is different from the counterpart of the line-follow part. In the line-follow case, the heading PID and the speed PID are cascaded, but the inputs (measurements) are from two different elements: the line sensors and the encoders, respectively. The following code snippet shows how the outer PID produces credible demands for the speed PID:

```
void straight_line(float input_bias, unsigned long interval, float angle,
 ↪  float measure_angle)
{
  /* input_bias: the constant speed | interval: time interval for
   ↪  straight-line PID | angle: demand of straight line=0 | measure_angle:
   ↪  measured orientation
  */
  unsigned long straight_pid_timestamp = micros();
  unsigned long straight_pid_time_interval = straight_pid_timestamp -
   ↪  straight_pid_prev_timestamp;
  if (straight_pid_time_interval > interval)
  {
    straight_output = straight_PID.update(angle, measure_angle);
    l_power = input_bias + straight_output;
    r_power = input_bias - straight_output;
    power_constrain(constrain_bias);  // Avoid low values that does not
     ↪  launch the car
    straight_pid_prev_timestamp = straight_pid_timestamp;
  }
}
```

One of the advantages lies in **anti-interference** performance. The car can reduce the influence of the oscillations in the wheels or the uneven surfaces of the map. Compared to single-stage PID control in which the expectations for the speeds are constant values, the cascade involves the accumulative heading error of the linear motion by another PID. The expectations of the algorithm which **compensates the heading errors** are more reasonable and accurate. Therefore, the linear motion is notably more closed to the ideal straight line than that of the one-stage PID control. However, the cascade PID controller consists of two sets of PID parameters. The **tuning job is more time-consuming** than that of the other case. Once the car configurations like wheel separation or distance per count change, all the six parameters require a readjustment.

(b) *Why did you use the solution you did?*

I choose the algorithm mainly because the **single-stage PID controller cannot drive the car run in an ideal straight line**. Without the credible demands produced by the heading

PID, the orientation I observed from debugging using the serial monitor varies around 2°. However, with cascade PID control, the error is dramatically reduced at about 0.01°. As a consequence, the linear motion is almost the same as the ideal straight line from the humanistic perspective, contributing to better tracking performance. The only component that requires correction of the accumulative tracking errors is the system error.

(c) *What else did you consider?*

I have tried the *single-stage PID controller* for the straight-line motion as discussed above. If the car operates with the one-stage PID, there are non-negligible randomly distributed position tracking errors which occur everywhere, and the accumulative error can rise considerably. Although the car can run in a smooth motion and can successfully finish all tasks and make a come back. However, the final positions where the Romi cars stopped were random and unpredictable. Efforts were made to eliminate the system error that only compensate the errors to a low extent.

Before applying PID control to straight-line motion, I purely used the *direct method*: driving the motors with constant powers. This method turned out to be doubtlessly unsuitable since the difference in two motors were magnified as time increased, causing a constant bias in orientation.

Another method I considered is to *compensate for the heading error of the last loop iteratively.* For example, the last loop witnesses a heading error at 2°. In the current loop, the car should move towards a slightly different direction to produce a heading error at −2°. However, I doubt that the method strongly depends on the low time interval, and the performance does not conform to the automatic control theory.

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.**

(a) *Identify your chosen problem in terms of the key elements of a robotic system: task, software, hardware and environment.*

Task. The problem I would like to investigate further is **robot localisation**. The specific task is to **drive a mobile robot through a maze** (whose details will be described below). The line-following task in this module assumes that the Romi car can be localised with an approximate dead-reckoning, i.e. by deciding the location of the next time step based on the previous pose and the velocity. Dead-reckoning, even coupled with the measured data of the optical shaft encoders, can not guarantee a low localisation error of the robot. The drift caused by the wheel slippage introduces most of the localisation errors, which will accumulate over time. Thus, I would study a more accurate localisation method which will be discussed. In the maze-crossing case, the car requires to decide the optimal path to walk through the maze when it is put and set up at the starting point, i.e. the entrance of the maze.

Software. This part consists of two main components: *Extended Kalman Filter (EKF)* with sensor fusion and the *Genetic Algorithm (GA)*. Most of the localisation algorithms depend on relating the environment characteristics. Therefore, I would predefine the landmarks of the map for the robot to collect data. EKF can combine the reckoned or the predicted localisation with the measurements from the sensors to produce a corrected value with less uncertainty than both two sources. The advantage of EKF is that it can solve non-linear problems by a linearisation at each state. GA aims to find the optimal solution among the current solution space. In maze-crossing problem, GA will be used to initialise a set of paths in binary-string forms then apply crossover and mutation to generate new paths. The fitness function is designed to evaluate the length of the path on which the robot can travel and whether it can get out.

Hardware. The mobile robot will be a robot car with two optical shaft encoders (for velocity measurements and pose reckoning), two motors and two wheels (the actuators of the robot), four ultrasonic sensors (the extra sensors to provide measurements for the EKF). The ultrasonic sensors are installed to the robot: two for the front-behind line and the other two for the sides. Distances to the known landmarks are measured to provide a location in the current map.

Environment. The map is a maze in which the pathways are separated by walls. The walls all continuous and do not leave a breach which the robot cannot pass through.

(b) *Discuss how you would design a study and how you would evaluate performance of a solution.*

I would decompose my study into three assessments. *1) Robot movements and measurements.* This part will mainly involve how to understand the basic concepts of motor actuation, velocity measuring, closed-loop control. The robot should be capable of running to the desired position from the current position in stable linear motion. *2) Odometry and localisation.* This part draws connections between ideas from different sources such as sensor fusion, probabilistic robotics, control theory. One should learn how to derive the location from the ultrasonic data that relates to predefined landmarks and design an EKF to make estimations from previous states and correct them with measurements. *3) Genetic Algorithm on maze exploration.* This part is slightly separate from the previous ones because GA operates before the robot runs. Thus, this part recalls and implements the basic concepts of evolutionary computing. One should find derive an optimal path that leads the robot out of the maze.

The evaluation should be reasonable from both the *humanistic* perspective and the *robot* perspective. For semantic reasons, the solution should be successful, accurate, jam-proof and reproducible. Concerning these features, one should conduct experiments to perform tasks, including obstacle avoidance, low oscillations, swift motion. For the robotic system, one must assure that the qualities of essential measurements are higher than the standard thresholds. Several examples are: the localisation errors derived by the EKF are low, or the optimal path derived by GA is practically executable, or the heading errors for the straight-line motion is negligible, etc.

(c) *Even though the Romi is a relatively simple robot, you have experienced some of the hidden complexity in using it to solve the line-following task. Discuss how your experience with the Romi influences your approach to your chosen robotics challenge.*

One tip for the robot localisation challenge is the **design of the system architecture**. As discussed above, designing the workflow of the Romi car did confused me for some days. The transfer from the prototype (the line-following task) to the whole robotics system (Romi car finishing the map) is tricky and challenging. I would stick to the principles in the study of the maze-crossing case or any other future situations that 1) design a programming-friendly workflow for the system and 2) always work out a prototype of a subsystem via experiments and then integrate them step by step. Therefore, I would first determine the robotic structure of the system, i.e. how the mobile robot finish all the tasks in the workflow. It would also be useful to consider questions including: ”are some of the subsystems cascaded?”, ”are some of the subsystems in a nested way?”, ”are there fixed order that the subsystems should follow?”... Therefore, the three assessments introduced above should be studied separately and connected reasonably.

The second tip is for **programming**. I would desire to split the operation of the mobile robot into several stages, like the solutions to the line-following problem. The switch-Case structure presents better software modifiability and more debugging convenience. Other experiences in programming like how to make use of sensor values shown in the serial monitor also guide me to use debugging information to detect problems in the software. For example, if I encountered a problem in measuring the location of the mobile robot, I would make the program print the four values read from the ultrasonic sensors and track the changes in variables until the computation of positions.