ELEN90066 Embedded System Design

Kobuki robot Project

Wednesday 2:15 – 5:15

Group 5

Yang Cao 751384

Zijie Yang 825694

Weihan Su 797680

ABSTRACT

The era of industrial revolution 4.0 is here, which introduces a smart factory concept. It leads to a large number of automatic robots to be widely used across the world. To keep sticking to the latest technology trend, knowing the mechanism and implementation skill of the embedded system are critical. The Kobuki robot can be regarded as a simple prototype of a modern intelligence vacuum cleaner android, which represents the most relevant consumer product. One objective is to use Kobuki to get in touch with the control theory and logic designing from the lower level for the future sophisticated robot designing in the career or the industry. This project brings ideas of ways for thoughts and implementation. Begin with the detailed analysis and follow with careful design and repeated test; finally, actual development can be achieved. By accessing the data from the 3-axis accelerometer and multiple sets of bump sensors and IR cliff sensors, the simple logic model can be created for the robot to overcome obstacles and to be able to deal with different operating situations safely, fluently and steadily. To test the performance and the response of Kobuki, virtual simulation, theoretical validation and the actual experiments were conducted and followed by continuous fine tunings. With the assist of the finite state machine, a general view of the processing logic can be easily observed. In the future, based on experiments and achievement, more advanced robots can be constructed. More precise and multifunctional sensors and enhanced logics will be deployed on the prototype to achieve a technological upgrade and a higher adaptation in varied environments.

CONTENTS

1 Introduction 1

1.1 Background 1

1.2 Purpose 1

1.3 Equipment 1

1.4 Sensor data: 2

2 Design 3

2.1 Specifications 3

2.2 Design procedure: 3

3 Validation 6

4 Testing & Adjusting 7

4.1 Simulations in the CyberSim 7

4.2 Reality experiment 8

4.2.1 Case 1 – avoid obstacles (confront directly on the ground) 8

4.2.2 Case 2 – barrier wall avoidance 10

4.2.3 Case 3 – redirection (uphill) 11

4.2.4 Case 4 – cliff avoidance 12

4.2.5 Case 5 – platform behaviour 13

4.2.6 Case 6 – redirection (downhill) 14

4.2.7 Case 7 - STOP 14

5 Final FSM 15

6 Outcome of the final workshop run 16

7 Discussion & improvement 17

8 Conclusion 18

References 19

Table of Figures 20

1 INTRODUCTION

1.1 BACKGROUND

With the rapid development of technology, it is possible to use the latest technology to help people deal with things effectively. For example, we are using a robotic vacuum instead of people to do housework. The robot can clean the entire room on their own for a higher time efficiency (IO, 2018). As early as 1996, people tried to manufacture similar machines to save the workforce of cleaning (Invention of RObotic Vacuum Cleaners, 2019). However, the problem of technology and cost limited them until the automated sweeping robot was popularized in 2004 (Bloomberg, 2013). Nowadays, the advances in technology makes fully automated sweeping robots achievable with a suitable price. This dramatically reduces the pressure on the housework and beneficially improves our happiness in life within a reasonable budget.

1.2 PURPOSE

This project is aimed to control a Kobuki robot without the vacuum cleaner to complete motions as a typical sweeping robot needed to do by using myRIO with a suitable algorithm to analyse the sensor values from Kobuki. That means the robot should be able to drive directly on the ground with obstacle avoidance, climb up and down a hill with cliff avoidance and stop on the ground within a required distance. The finite-state machine helps us to construct the whole algorithm. Meanwhile, it is useful to use CyberSim to simulate the algorithm virtually before the actual testing, which saves much time on Eclipse setup and executable file transferring.

1.3 EQUIPMENT

MyRIO-1900 is a real-time embedded evaluation board made by National Instruments which comes with an ARM and an FPGA processor. It has multiply reconfigurable I/O port, including two myRIO Expansion Port (MXP) connectors (myRIO-1900 User Guide and Specifications, 2016). One of them is connected to the Kobuki robot. Kobuki is a mobile research base made by Yujin Robot. This robot has multiple sensors and multiple IO interfaces and, more importantly, it has an RS-232 serial port which can be used to connect to the myRIO (Online User Guide – KOBUKI, 2015). As the controller of Kobuki, MyRIO runs a real-time system. It communicates with Kobuki by using predefined protocol via a Kobuki driver (Daniel Stonier, 2017).

In this project, we first use cross-compilation in Eclipse to compile the source code, including main state chart, Kobuki driver, and myRIO driver into a single ARM executable file. Next, transfer the executable file to myRIO by using the sftp protocol. By running the program through ssh protocol on myRIO using the

nohup command, myRIO will ignore the HUP (hang-up) signal, which reduces the risk of stopping the program while Kobuki is running. When the program is running, the program reads the data in myRIO and Kobuki sensors via the driver and makes judgments based on the design of our statecharts.

1.4 SENSOR DATA:

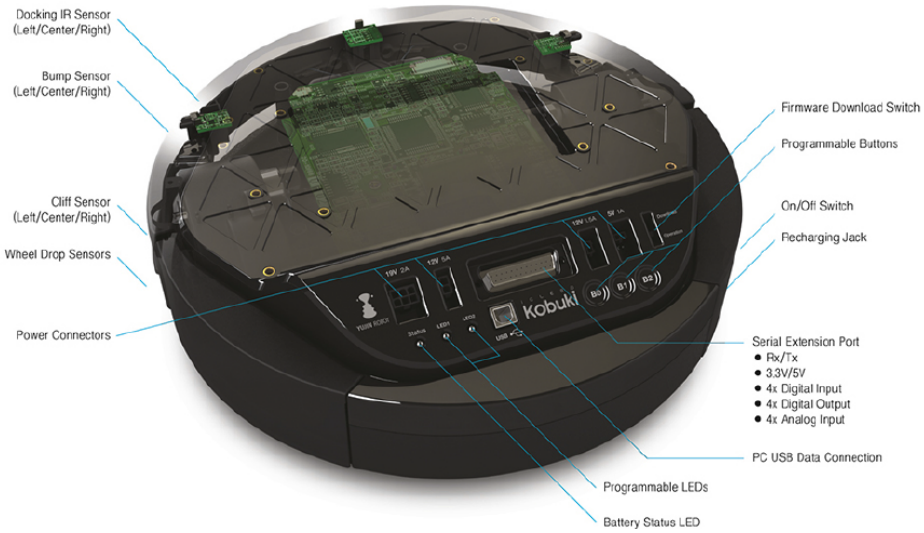


Figure 1: Kobuki sensors' locations

Figure1 above shows the sensors, buttons and ports distribution on the Kobuki robot (Online User Guide – KOBUKI, 2015). There are several essential sensors that we need to pay attention to:

1. **Button sensor (sensors. buttons.B0)**

   This button sensor value is used to control the robot's movement. By pressing this button, the robot should react with play/pause/resume differently.

2. **Bump sensors (sensors.bumps_wheelDrops.bumpLeft & sensors.bumps_wheelDrops.bumpRight & sensors.bumps_wheelDrops.bumpCenter)**

   These three bumper sensors on the front of the robot are used to detect the unwanted obstacles and provide three Boolean values output. Whenever these sensors detect non-zero values, it means that the obstacle appears, and the obstacle avoidance algorithm should be applied.

3. **Cliff sensors (sensors.cliffCenter & sensors.cliffRight & sensors.cliffLeft)**

   The cliff sensors on Kobuki are three IR sensors which measuring distance between bokuki and floor. A sudden increase in the distance means there is a cliff under the sensors. The driver of Kobuki reads the raw data of IR sensors and provide three Boolean values output to indicate the existence of the cliff. These three cliff sensors located around the robot. To avoid falling off and wheel drops, whenever these sensors detect non-zero values, the cliff avoidance algorithm should be applied.

4. **accelerometer which is located on the myRIO (accelAxes.x & accelAxes.y & accelAxes.z)**

   The myRIO onboard accelerometer is an MMA8452Q 3-axis digital micromachined accelerometer made by Freescale (Jeff C. Jensen, 2015). The accelerometers measure data by detecting the inertia of

a movable mechanical structure and provide a 12-bit output of acceleration data. These three accelerometer's different axis readings are beneficial. They can not only be used to detect the incline, but also be used to maintain the required direction during the hill climbing.

2 DESIGN

2.1 SPECIFICATIONS

1. By pressing 'B0' button to power on, stop or resume the robot.

2. The robot should drive directly forward and avoid an obstacle on the level ground with ground orientation, which is the direction pointed by the robot's front after pressing 'B0' button.

3. The robot should climb the hill without falling off from cliffs and avoid the obstacles during climbing. When the robot achieving the top of the ramp, keeps driving through the top flat section. Finally, go down the downward ramp and stop the robot after 40cm of the hill bottom.

4. The robot should never rotate more than 180°.

5. The robot should not act chattering nor obstacle hugging.

6. The total time to finish driving, climbing and finally stopping should be no more than three minutes.

2.2 DESIGN PROCEDURE:

1. In the workshop 4, we designed the navigation part. To ensure that the robot could avoid the obstacles on the level ground, we let the robot go back firstly whenever the bumper sensors detect obstacles. Then always let it turn right with 90 degrees and be followed by a directly forward driving with 25cm. After that, turn left with 90 degrees, which led the robot back to the ground orientation direction and keep driving. This works for simply situations. While it did not work for the situation that there is a wall on the righthand side and an obstacle in front of the robot at the same time. While there was still another problem: When the robot hit an obstacle, there would be a slight change in its direction. This led to a mismatch of the ground orientation specification.
2. In the workshop 5, we dealt with the hill climb problem:

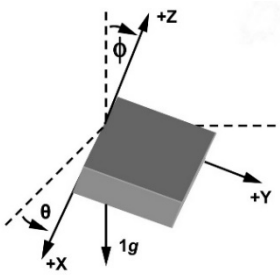


Figure 2 Angles of Spherical Coordinate System

Firstly, according to the 3-axis accelerometer datasheet, the equation was found to represent the angle in the z-direction: $\phi = \cos^{-1}\left(\frac{accelAxes.z}{\sqrt{accelAxes.x^2+accelAxes.y^2+accelAxes.z^2}}\right)$ (Fisher, 2010). Figure2 illustrates the relationship between the ϕ angle and the 3-axis coordinate. Therefore, the simplified formula $\frac{accelAxes.z}{\sqrt{accelAxes.x^2+accelAxes.y^2+accelAxes.z^2}}$ could be used to detect the incline. When the formula's value was smaller than a threshold, the robot was indicated on the ramp. The threshold needed to be found by several simulations and tests in reality. Secondarily, the algorithm was written to ensure the robot would go back with 20cm and then turn around when the centre cliff sensor detects the cliff at the top of the ramp to avoid the wheels drop. Additionally, the robot movement needed to be reorientated on the ramp, if the starting point of the hill climb section is not directly pointing to the ramp or the robot did not move in a straight line on the ramp. This was achieved by keeping reading the x-axis accelerometer sensor values and changing the robot with 5 degrees when it did not drive in a straight line. The threshold that indicated the robot's wrong movement was tested by multiple times simulations and should be 0.02. Besides, cliff avoidance is important, as well. To ensure neither the wheel nor the whole robot would not fall off from the ramp, the robot should have similar reactions with the obstacle avoidance. However, at this time, the three cliff sensors should be used instead of the bumper sensors. Finally, the robot went downwards from the ramp top and stopped after it reached the level ground.

3. In the workshop 6, we used statecharts in the LABVIEW to control Kobuki, instead of using the c code. It was easier to debug by using the statecharts because we could see the current running states directly in the CyberSim. However, it was not convenient to construct statecharts in the LABVIEW. Also, to minimise the variables used in the LABVIEW, we constructed much more states in the statecharts than that in the c code FSM. Thus, the whole chart was quite complicated. In conclusion, we decided to use c code to design the final project.

4. In the workshop 7 to 9:

   The final project requirements are slightly different from the previous workshop exercises, so we needed to make some modifications.

   First, the obstacle avoidance should be improved to deal with the problems that cannot be solved by keeping turning right. We designed a "U-Turn and Across" algorithm to resolve the situation that there is a wall on the righthand side and an obstacle in front of the robot. The FSM of this algorithm is shown in Figure 3. The first part of this algorithm is the same as what we did in workshop 4, which goes back and turn right when the robot detects obstacles. However, after the robot turn right, if there is a wall on the righthand side, the robot will go straight and hit the wall. The point of this algorithm is, after the robot detects the righthand sidewall, it will go back and perform a U-Turn and go straight until it detects the left-hand side wall or another obstacle. After this "across", the robot will turn right and back to the normal driving state. The drawback of this algorithm is that the robot

may spend more time in the "across" part. Moreover, since the travel distance of "across" may be far, it will cause a big problem if the robot has an angular deviation from the ground orientation.

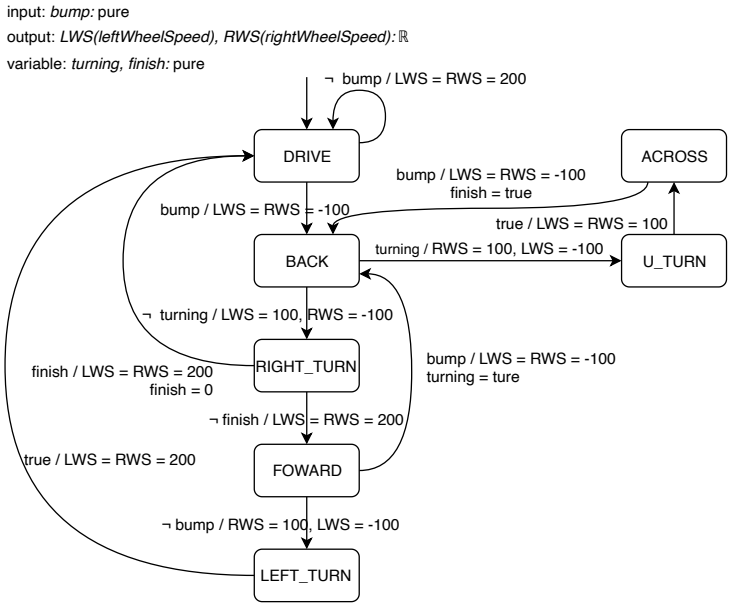


Figure 3 FSM of “U-Turn and Across”

Therefore, we designed the second “Switch Avoidance Direction” algorithm with a more robust wall avoidance mechanism. The difference between two algorithms is that, after the robot hit the righthand sidewall, it will enter a state with an opposite obstacle avoidance direction to its default state, which is keeps turning left. The FSM of this procedure is shown in Figure 4. In order to reduce the complexity of the code and state, we use a flag to represent Right Turn State and Left Turn State.

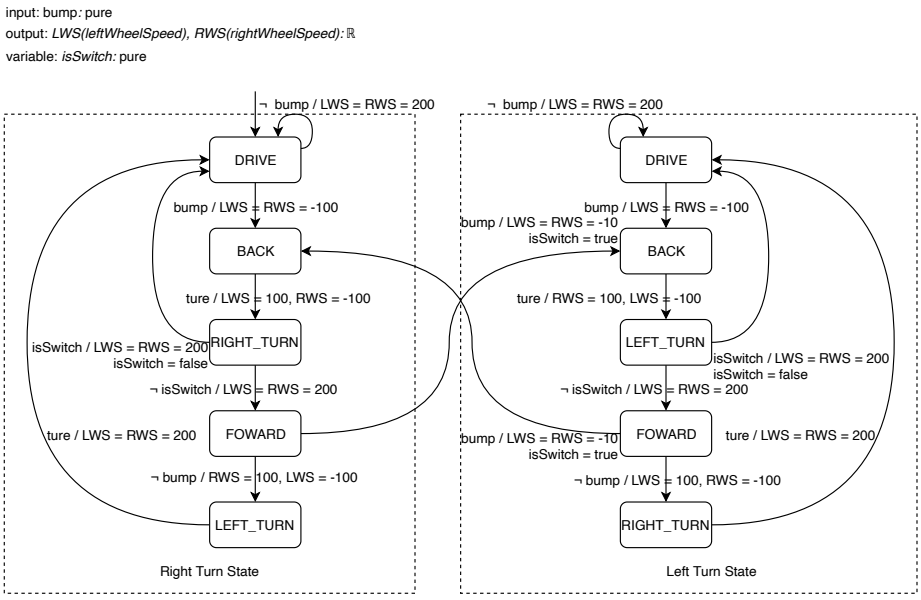


Figure 4 FSM of “Switch Avoidance Direction”

In our process of testing obstacle avoidance algorithm, we realized that when the walls on the left and right sides are not parallel to the forward direction of the robot, the robot may fall into an infinite loop.

When the right-hand side wall is detected, the robot may continually avoid the wall but cannot switch from the right turn mode to the left turn mode. To prevent this from happening, we changed the travel distance of the FOWARD state to make it easier for the robot to switch from right turn mode to left turn mode.

Secondarily, in the previous workshop, our robot only needed to go downhill directly after climbing. In the project, our robots need to go through a flat platform after climbing and finally go downhill. Since the whole process is deterministic, we have designed a one-way algorithm to ensure robustness.

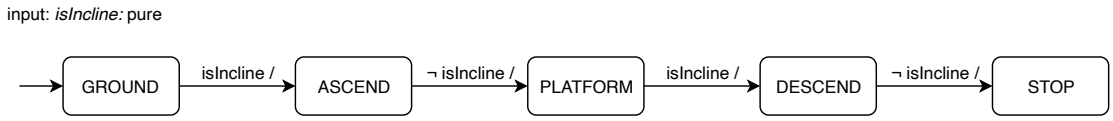


Figure 5 FSM of Hill Climb

The FSM of hill climb is shown in Figure 5. The robot uses accelerometers to detect the slope by computing slope coefficient $\frac{accelAxes.z}{\sqrt{accelAxes.x^2+accelAxes.y^2+accelAxes.z^2}}$. If the result value is smaller than the threshold, it can be determined that the robot is on the slope. After several tests and validation, we finally set the threshold at 0.99. In order to reduce the complexity of the code and the number of states, we use flags to indicate ascend, platform and descend.

Finally, Kobuki should stop within 40cm when the bottom of the slope is reached. Since the 40cm is an upper limit, we decide to set the stopping distance at 100mm to leave enough space for Kobuki to detect the ground. The threshold we used for detecting bottom of the slope is the same as the threshold we used for detecting slope. If the slope coefficient is larger than the threshold 0.99 after hill descends, Kobuki should stop within 100mm.

3 VALIDATION

After the rough design, there are several critical specifications that our model needs to satisfy. According to the temporal logic theorem learned from lectures the reachability analysis could be done as shown below:

The first one is obstacle avoidance. The Kobuki robot should be able to start a certain obstacle avoidance method at any time if it is in Drive state, and any of its bumper or cliff sensors are triggered. This property can be represented by using temporal logic as

$$\begin{aligned}&\boldsymbol{G}\big(DRIVE \wedge (anyBumpSensors \vee anyCliffSensors)\\&\wedge\ \boldsymbol{X}\big(\ BACK \wedge (turn\_direct \wedge LEFT) \vee (\neg turn\_direct \wedge RIGHT)\big)\\&\wedge \boldsymbol{X^2}(FORWARD) \wedge \boldsymbol{X^3}\big((\neg turn\_direct \wedge LEFT) \vee (turn\_direct \wedge RIGHT)\big)\\&\Longrightarrow \boldsymbol{F}\ DRIVE\big)\end{aligned}$$

By performing an execution trace on the final FSM (Figure 23) starting from the Drive state with any bumper or cliff sensor are triggered, the result exactly follows our logic formula and finally leads to the Drive state. Therefore, it can be proved that our model satisfies the obstacle avoidance property.

The second critical property is that, if any of Kobuki robot's bumper or cliff sensors are triggered at Forward state, our model should be able to switch the avoidance direction as we designed. This property can be formulated as

$$\boldsymbol{G}(FORWARD \wedge turn\_direct \wedge (anyBumpSensors \vee anyCliffSensors) \Longrightarrow \boldsymbol{F}\neg turn\_direct)$$

As can be verified from the final FSM (Figure 23), at Forward state, any of bumpers or cliff sensors will trigger a reversion of the avoidance direction which indicates that our model satisfies this property. Thirdly, our model should satisfy the properties

$$\boldsymbol{G}(\neg ascend \wedge \neg descend \wedge \neg platform \wedge (inclination\ coefficient \leq 0.99) \Longrightarrow \boldsymbol{F}\ ascend),$$
$$\boldsymbol{G}(\neg ascend \wedge \neg descend \wedge platform \wedge (inclination\ coefficient \leq 0.99) \Longrightarrow \boldsymbol{F}\ descend),$$
$$\boldsymbol{G}(\neg ascend \wedge descend \wedge \neg platform \wedge (inclination\ coefficient \geq 0.99) \Longrightarrow \boldsymbol{F}\ STOP)$$

at any state. It implies that Kobuki should be able to distinguish slope and ground and perform the corresponding action. As can be seen from the final FSM (Figure 23), any transactions from the Drive state have rigid guards which guarantee the detection of a slope. The ascend and descent flags can be set to present only after detecting a slope at ground and platform respectively. Also, our model enters the Stop state only if descend flag is present and ascend, and the platform flag is absent, and inclination coefficient is larger than 0.99. Therefore, our model satisfies the properties of slope detections. In summary, the designed algorithm satisfies all the critical specifications in theory.

4 Testing & Adjusting

4.1 Simulations in the CyberSim

Before implementing our algorithm on the real Kobuki robot, it should be simulated in the CyberSim first to check whether it works or not. Because it took time to set up Eclipse for code compiling, transfer it to myRIO and test under the required environment. The CyberSim simulations could save our time to find out any algorithm errors.

1.

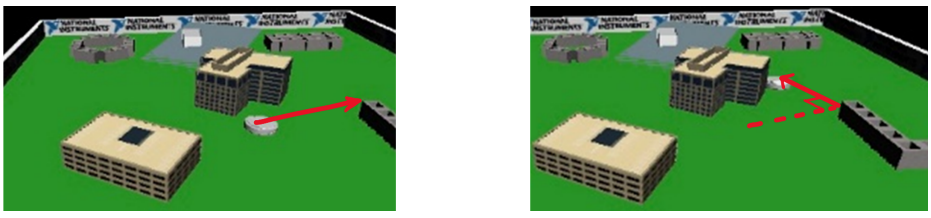


Figure 6 Obstacle avoidance simulation

The robot can avoid obstacles on the level ground that was verified via simulations (Figure6). The robot detected the front obstacle and turn left to avoid it.

2. We found that the x and y-axis in the simulation are opposite to that on Kobuki in reality due to the different direction of the myRIO placement. Thus, we needed to exchange the analysing acceleration axes for the inclination detection and reorientation procedure on the ramp. In addition, the threshold that used to detect the ramp is quite difficult to found. We tried several values and finally found that 0.99 was suitable under simulation conditions.

   The robot can detect the ramp and reorientate to the straight-line during inclination (Figure 7).



Figure 7 Inclination detection and redirection simulation

3. The robot can avoid obstacle correctly on the ramp which is illustrated in Figure 8.

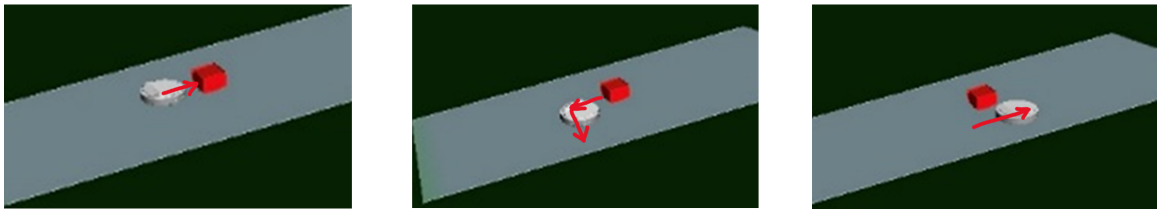


Figure 8 Obstacle avoidance on the slop simulation

4.2 REALITY EXPERIMENT

In actuality, the behaviours may be different from the simulation, because the simulation is under a relatively ideal situation, while the reality is not. Therefore, the actual tests are necessary to fine-tune the sensor threshold and critical parameters for stable processing.

To achieve a better performance in the final assessment experiment, some cases that the robot may encounters were built. To analyse which state the robot has entered, we designed to use a STOP state to detect the behaviour, which is visualized.

4.2.1 Case 1 – avoid obstacles (confront directly on the ground)

Based on the design and the simulation, Kobuki will start from a plane surface (ground), it is expected to keep moving straight at speed 200 units (default maximum is 200) until meets the obstacles. When the bump sensors are activated the first time, the robot should back 150mm with maximum speed, then turn to the right in situ (90° clockwise) with speed 100 for the left wheel and -100 for the right wheel, and move forward for 250mm with speed 200 if there are no other obstacle encounters, then turn left (90° Counter-Clockwise) with the left wheel at speed -100, and the right wheel at speed 100, afterwards,

Kobuki should back to the Drive state, drive straight. If the robot hits another obstacle after the first turning and before it finishes the 250mm path, which can be considered as the robot is facing a right corner. In this situation, the design will guide the robot get back for 150mm first, then perform the inverted behaviour, turn to the left and keep detecting the obstacle. If the hit is detected, then back first to reserve a free turning space, then turn left and repeat the actions as mentioned before to bypass the obstacle, the ideal final angle should be the same as the facing angle at the start point.

A multi-situation map was constructed for this test (Figure 9)

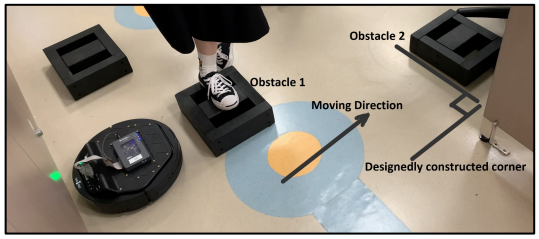


Figure 9 Obstacle avoidance test map

As the expectation, Kobuki went straight and hit the first brick, the bump sensors detected a bump event, then the robot backed ~150mm, and turned right 90° smoothly, so far, the system operated smoothly and stably, which confirmed the speed for the Back state and the Turn state were suitable. Subsequently, Kobuki ran ~250mm along the straight line, and the process followed by another left 90° turning to keep the original facing angle (Figure 10). The robot returned to Drive state by now.

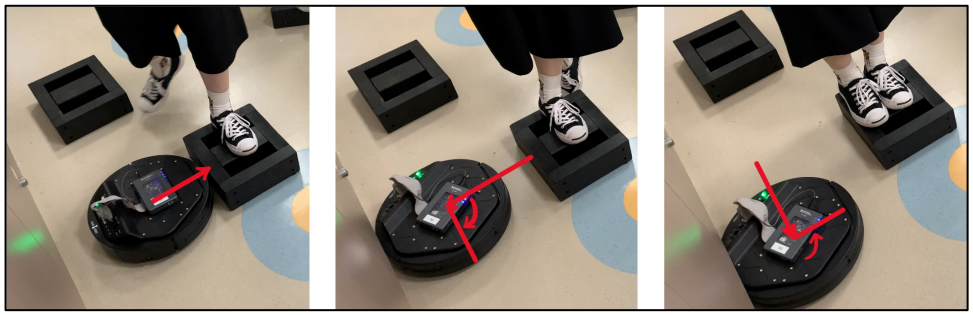


Figure 10 Obstacle avoidance test illustration- set 1

Since it hit the Obstacle 1 again, it repeated the previous action as we designed. Kobuki successfully bypassed the 1st obstacle and kept the angle as it starts (Figure 11).



Figure 11 Obstacle avoidance test illustration- set 2

After a straight running, the 2nd obstacle was met, the process followed the previous one until Kobuki hit the desk side during the Forward state. It recognized the corner situation correctly, and started performing an inverted operation, kept a left moving tendency (Figure 12). In Figure 13, the right bump sensor was activated, and the robot backed immediately and accurately behaved, which led Kobuki surpassed the Obstacle 2 finally.



Figure 12 Obstacle avoidance test illustration- set 3



Figure 13 Obstacle avoidance test illustration- set 4

Via the test, our control logics and critical parameters for Kobuki to avoid the obstacle on the ground were verified to be satisfied with the specifications.

4.2.2 Case 2 – barrier wall avoidance

A particular situation is needed to be considered, according to the project information, our Kobuki will face a map as figure 14 shows, the circled place is tricky. Based on the obstacle avoidance logics, when Kobuki enters this region, it must change its moving direction oppositely, to realize this process, it requires the robot to hit the sidewall during the Forward state. If θ is small. Assume θ is $20°$, in this situation, the relationship between the distance for Forward state and that for Back state can be calculated, $\tan(\theta) = \frac{Distance_{Back}}{Distance_{Forward}}$, suppose the forward distance is fixed, 250mm, then the maximum distance for Back is 91mm. If the back range is larger than the threshold, Kobuki will require extra total back distance and more repeated detection to change its direction until it reaches the sidewall, since the robot will never enter an encounter stage during Forward state, which will waste plenty of time. Therefore, the back distance was set to 80mm to save time.

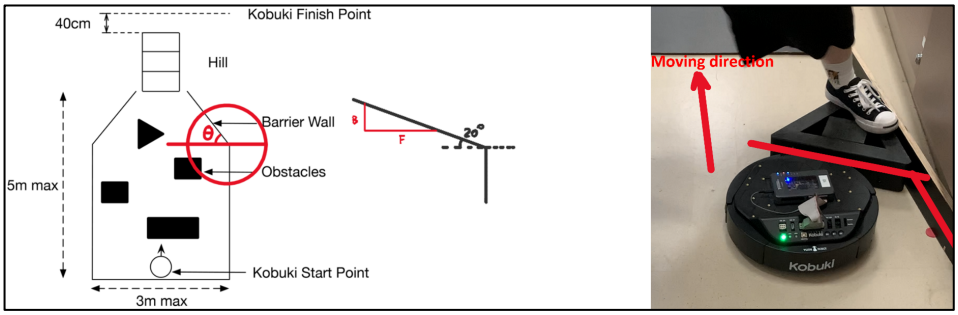


Figure 14 Barrier wall avoidance test map

Due to the equipment limitation, we were not able to use an obstacle that is long enough to test this logic with defined parameters. Instead we only test the turning mechanism by using a small triangular brick.

After the collision occurred, Kobuki did the avoid obstacle flow correctly; once the second collision happened during the Forward state, it altered its direction oppositely and followed the logics strictly. This situation can be overcome as Figure 15 shows.

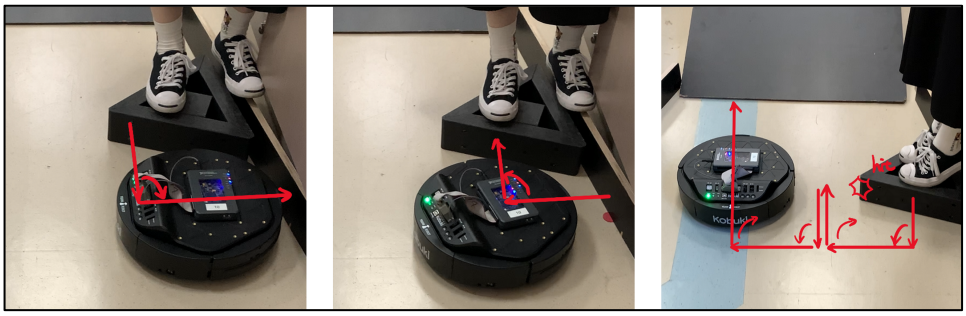


Figure 15 Barrier wall avoidance test illustration

4.2.3 Case 3 – redirection (uphill)

Adjusting the threshold for the hill detection logics is essential, with an error threshold, Kobuki will never enter the designed “ascend” state correctly. Firstly, 0.99 was used as the threshold, which was obtained in the LabView simulation, the test result was successful. However, this value was not perfectly stable in the actual implementations every time and misreads could happen randomly. Sets of values around 0.99 were tested afterward, as a result, 0.99 was still the one with the best stability to detect the inclination among all tested values.

Initially, Kobuki would spin a certain degree to maintain its position straight to the top of the slope, via multiple tests. However, it could be noticed that the body was not stable, and vibrated a lot, due to the frequency angle detection and large speed differences between the moving status and turning status. Therefore, we had adjusted our logics, a smooth redirection requires the wheel speed differences to be moderated, while too small may take more time to recover the angle. Again, through multi-times of experiments with the consideration among the time efficiency, the turning efficiency, and the stability, the logic was decided as keeping one wheel remains its original speed, and slightly decreases another wheel

speed, the difference between them is 50 units (200: 150). Under such situation, Kobuki could ascend the hill smoothly and steadily.

For the test (Figure 16), Kobuki was placed with an angle, which created an initial bias. Once the calculated value from the accelerometer reached the threshold, the robot started to compare the current angle and the right angle, if the difference between them is larger than 5°, then it would be followed by a turning process, and keep tracking the angle difference for balancing the straight position to the hill. Finally, Kobuki could arrive the top of the hill as the expectation (Figure 17).

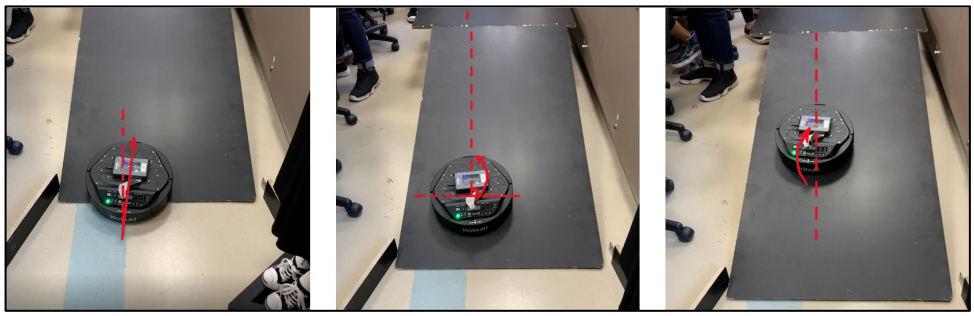


Figure 16 Inclination detection and redirection test illustration- set 1

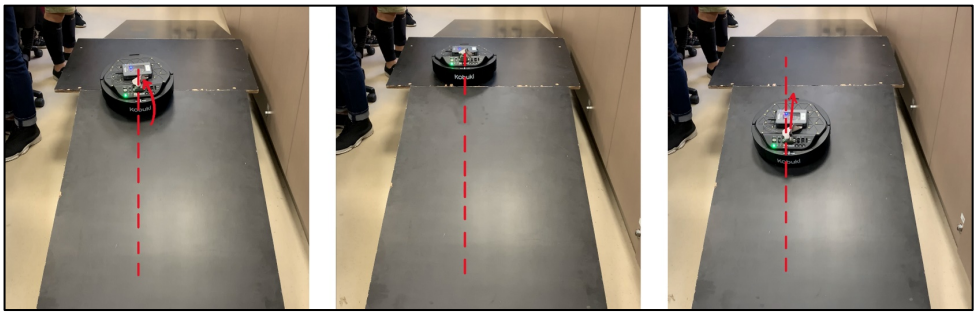


Figure 17 Inclination detection and redirection test illustration- set 2

4.2.4 Case 4 – cliff avoidance

During the inclination, cliff may be encountered, to avoid an unexpected falling, an avoidance mechanism should be introduced. The designed logics show a procedure will get involved, which is the same as the obstacle avoidance on the ground to bypass the cliff, and back to the right path. It could be found that a cliff meeting event can occur during the turning process or the straight moving status due to the physically unbalanced wheel speed, which we were not able to avoid.

Some intended cliff detection events had been created.

1. Placed Kobuki near the corner at the top of the hill (Figure 18)

   In this situation, the cliff sensor was activated before Kobuki reaches the top, based on the designed process, the robot followed the obstacle avoidance rules, backed and then turned around. Kobuki entered a corner situation as that been tested in the section 3.2.1. Finally, it succeeded to reach the top and pointed straight.

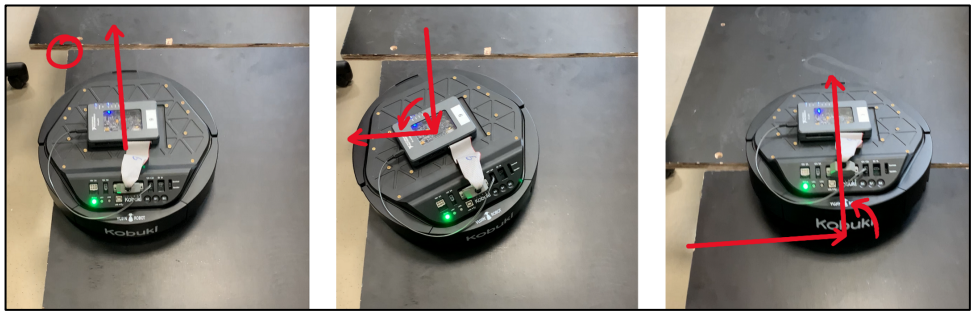


Figure 18 Cliff detection and avoidance test 1 illustration

2. Placed Kobuki at the bottom of the hill and closed to one side (Figure 19)

   Designedly, Kobuki encountered a cliff before a successful ascend detection, avoidance logic was involved. The robot behaved ideally as the expectation, which proved the cliff avoidance logic is robust and feasible during the whole operation flow.



Figure 19 Cliff detection and avoidance test 2 illustration

4.2.5 Case 5 – platform behaviour

After reaching the top of the hill, Kobuki should be able to detect the change in the inclination and recognise that it is entering a platform via the accelerometer. Within this region, the robot should maintain its orientation as the direction when it reaches the top, and only the redirection function should be disabled, the rest functions should keep functioning until passing the platform region, the redirection function is on again.

To test this case, the robot should start from the ground and climb the hill, then enter this region, according to the detection logic, the sequential order must be strictly followed, and we only focused on the platform region in this section.

1. Test without interference (Figure 20)

   After climbing, and reaching the top, Kobuki was able to detect the platform correctly, and kept moving with the same direction and passed the platform directly, since the direction had been appropriately adjusted during the ascent.

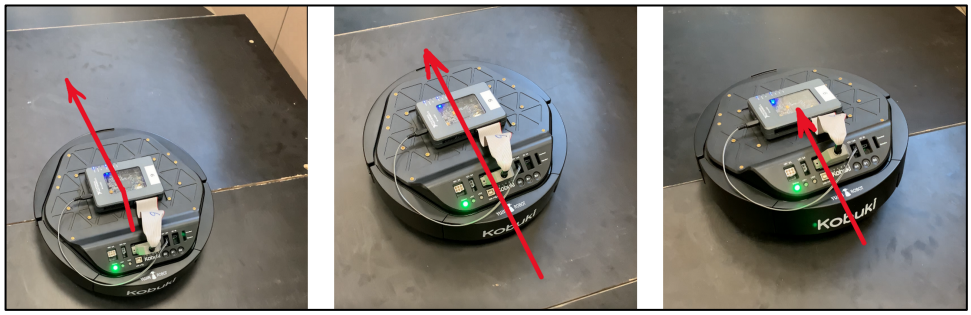


Figure 20 Platform behaviour test illustration

2. Test with interference

   During the test run, our Kobuki was able to detect the collision from another robot or detect an unexpected cliff if the driving direction was biased, and started the avoidance correctly, which proved the logic is effective.

4.2.6 Case 6 – redirection (downhill)

After passing a platform, a declination will be met. According to the specification, remain a straight position is required. Based on the experiments, Kobuki had a slight trend to the left. This may attribute to a physical error of the wheels, even two wheels were set at the same speed level, the left one would always slower than the right wheel. However, it could be easily observed that the robot always tried to recover its original angle to keep the general movement in the correct direction, which proved the logic was controlling the direction in perspective as Figure 21 displays.

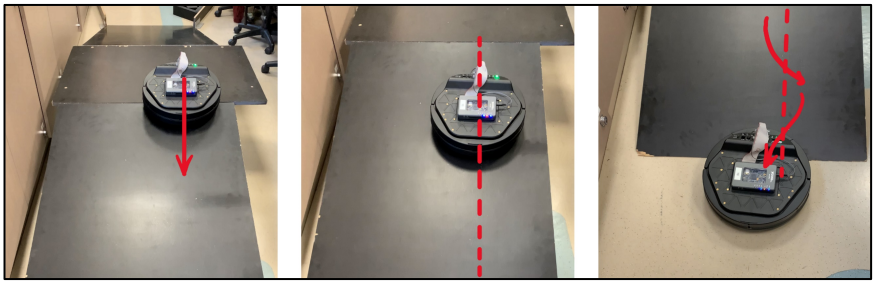


Figure 21 Downhill redirection test illustration

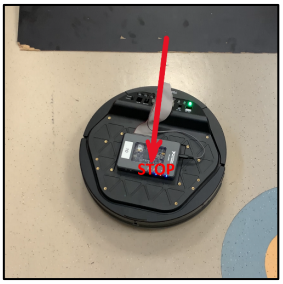


Figure 22 Stop state test illustration

4.2.7 Case 7- STOP

The specification requires a stop range within 40cm (400mm). Thus, a distance of 100mm was set. To identify the current status, a slower speed was set for the Final state.

The actual experiment showed Kobuki was able to perform correctly and stopped within the 40cm range with a ~10cm movement after leaving the slope (Figure 22).

5 FINAL FSM

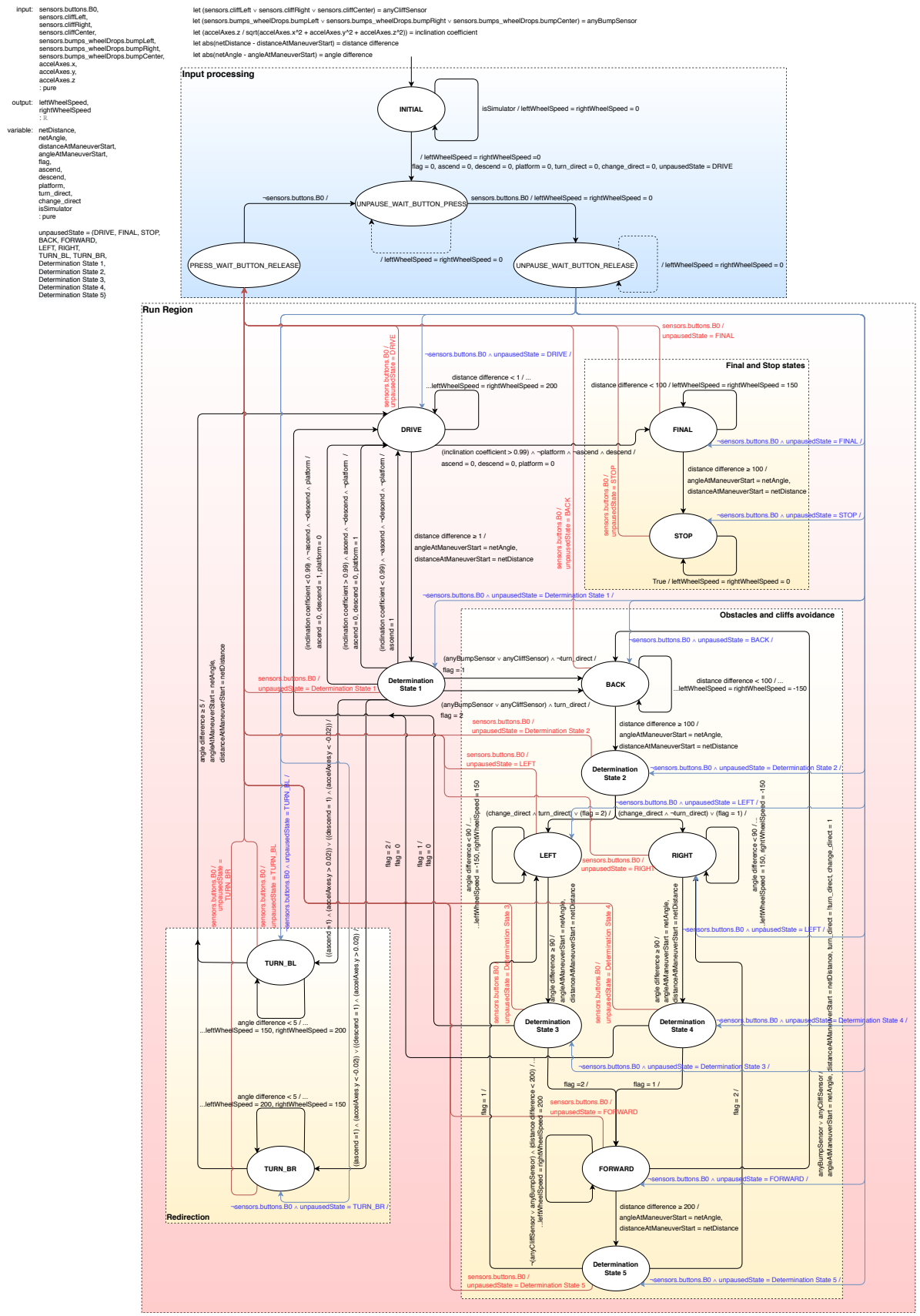


Figure 23 Final Finite State Machine

6 OUTCOME OF THE FINAL WORKSHOP RUN

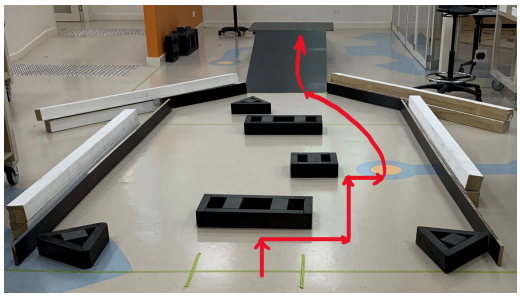


Figure 24 Final workshop map and actual operation route

The whole running time for the final workshop was 79 seconds. Our Kobuki robot could successfully avoid all the obstacles on the ground via the right route as the expectation (Figure 24). It avoided the first obstacle successfully and kept the starting orientation at first as Figure 25 shown below, while due to the friction difference between two wheels, the left wheel had a lower speed in the final run, the robot could not drive in a straight line, even both wheels were set at the same speed. Thus, the robot had a left trend when it intended to move straight.

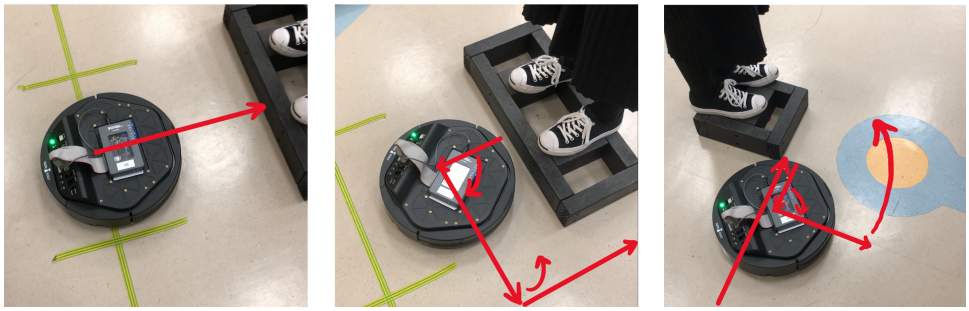


Figure 25 Final run illustration- set 1

An unexpected situation occurred due to this phenomenon after the robot avoided the second obstacle. It fortunately avoided the righthand side barrier wall without any impacts. Then the rest part was under control. Kobuki climbed up the slope with automatic angle correction to ensure it go straight (Figure 26). It should be noticed that the right wheel arrived at the top before the left one, which led the body had an undesired left turn and caused an extra cliff detection during the descent.

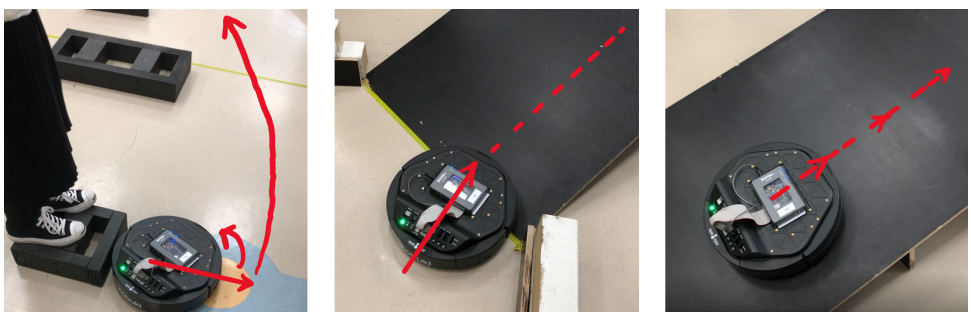


Figure 26 Final run illustration- set 2

After passing the platform, the robot started to descend the ramp and redirected the angle automatically. As mentioned, the latest posture triggered the left cliff sensor on the ramp, while it strictly followed our design and successfully avoided the cliff then went down the hill with activated angle redirection. Finally, Kobuki stopped within the required 40cm-distance after climbing down the ramp (Figure 27).

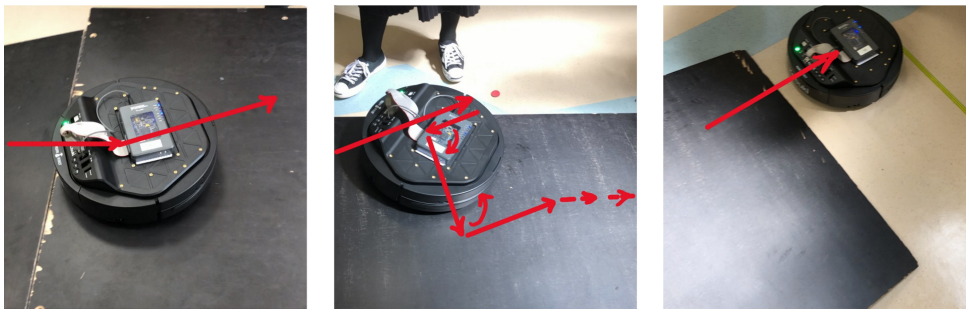


Figure 27 Final run illustration- set 3

7 DISCUSSION & IMPROVEMENT

During the whole project process, we had the opportunity to apply the knowledge that learnt in class. For instance, how to make sensors and actuators serve our algorithm, and the application of the finite-state machine in reality to control the robot's movements. This is very useful for us to understand theoretical knowledge through practical operations.

After all the design and implementations, all the specifications can be achieved. While there is still space for improvements, such as robustness. Due to the limitations of time, equipment and costs, it is difficult to deal with advanced improvement during this project. However, they can be solved in the future:

1. During the actual testing process, we found that there is a small difference in the rotational speed between the wheels on both sides of the Kobuki robot. This led the robot could not drive in a straight line even if the wheels on both sides were given the same speed. The cause of this error may be the difference in friction between the wheels and the axles. To deal with such issue, the voltage level of both wheels should be adjusted. If set two wheels at the percentage of the speed, but the robot bias to the left, which means, we need to decrease the percentage of the right wheel or increase the speed of the left wheel, repeated test, to find a balance point.
2. The algorithm mentioned above to avoid the obstacles is not sufficiently enough. If the first obstacle appears both in the front of and to the right of the robot, it will take a long time for the robot first to turn right and then realised the route on the righthand side does not work. Then turn left to avoid the obstacle. There is an optimal algorithm to save time. Once the left sensor detects an obstacle, turn right to avoid it, and vice versa. If the intermediate sensor detects an obstacle, take the same action as the left sensor detection situation does. While this algorithm is only optimal in theory. In reality, whenever the robot hits an obstacle, inertia and bump forces will cause the robot to change its angle

slightly. This may cause the robot to fall into an infinite loop of left and right turns. To solve such problem, there are two methods. Firstly, reduce the robot moving speed to decrease the inertia and bump forces so that the angle change can be minimised. Secondarily, replace the bumper sensors with that can accurately measure the distance from the obstacle, then the robot will be able to realize the obstacle before hitting with it. This can fundamentally solve the problem of the changed angle. However, both methods have their shortcoming. The first method will make the whole process very slow and will be unable to drive through the specified area within 3 minutes. For the second method, such precise sensors are expensive and are not conducive to reducing the costs. Since this robot will be sold as a commodity, the cost-efficiency is important. In conclusion, the trade-off is needed to improve this Kobuki robot.

3. Adding a gyroscope along with the existing accelerometer for measuring the angular velocity can achieve more precise steering and more accurate determination of the slope.
4. In order to meet the project requirements and improve the robustness, some algorithms are designed to be executed in single-time and one-way. However, in reality, robots may need to face complex situations, so a more intelligent state-switching algorithm design is needed.

8 CONCLUSION

In this project, the Kobuki robot is controlled by the myRIO with the designed algorithm in c-language. The same parameters and the same logic used in c- language design was applied to statecharts in the LABVIEW once. The result that we have found is that the efficiency of the statecharts in the LABVIEW is lower than the c-language, that is the reason why we chose C to control the robot. The whole algorithm was divided into five stages, including obstacle avoidance, carrier wall avoidance, climbing up the hill, driving on the flatform and climbing down the hill. Each stage was designed, simulated through the CyberSim, tested in the reality and improved separately and combined to achieve the final specifications.

Through this project, we learnt how to use myRIO to control the Kobuki robot and be familiar with the function of used sensors, such as accelerometers, bumper sensors and cliff sensors. In addition, our ability to construct and implement the finite-state machine in the c- language was improved. Meanwhile, the skill to identify errors and improve them in the c-language has also been enhanced. Lastly, our teamwork ability has been exercised, which is significantly helpful for our study and work. Moreover, this is also an essential ability for us to become an engineer in the future.

In conclusion, the appropriate price and fully functional automated sweeping robot are achievable. With applications of the improvements mentioned above, such a robot can be perfect in the future. To serve people, reduce their pressure and create an ideal living environment.

REFERENCES

Bloomberg. (2013, Oct 6). *How the Roomba Was Realized*. Retrieved from Bloomberg: https://www.bloomberg.com/news/articles/2003-10-05/how-the-roomba-was-realized

Daniel Stonier, Y. J. (2017, March 27). *kobuki_driver: Appendix : Protocol Specification .* Retrieved from kobuki_driver Documentation: https://yujinrobot.github.io/kobuki/enAppendixProtocolSpecification.html

Fisher, C. J. (2010). Retrieved from Using An Accelerometer for Inclination Sensing: http://LeeSeshia.org/lab/releases/1.70/documents/Fisher_10_UsingAnAccelerometerForInclinationSensing.pdf

Invention of RObotic Vacuum Cleaners. (2019). Retrieved from Vaccum Cleaner History: http://www.vacuumcleanerhistory.com/vacuum-cleaner-development/history-of-robotic-vacuum-cleaner/

IO, C. -C. (2018, DEC 05). *ROBOT VACUUM:ADVANTAGES OF SMART CLEANING*. Retrieved from CITI I/O: https://citi.io/2018/12/05/robot-vacuum-advantages-of-smart-cleaning/

Jeff C. Jensen, E. A. (2015). *Jensen, Lee and Seshia An Introductory Lab in Embedded and Cyber-Physical Systems.* Retrieved from University of California, Berkeley: https://ptolemy.berkeley.edu/books/leeseshia/lab/releases/1.70/EmbeddedLab_v1.70_RemoteDocs.pdf

myRIO-1900 User Guide and Specifications. (2016, May). Retrieved October 2019, from National Instruments: http://www.ni.com/pdf/manuals/376047c.pdf

Online User Guide – KOBUKI. (2015). Retrieved from Yujin Robot: http://kobuki.yujinrobot.com/wiki/online-user-guide/

TABLE OF FIGURES

Figure 1: Kobuki sensors' locations 2
Figure 2 Angles of Spherical Coordinate System 3
Figure 3 FSM of "U-Turn and Across" 5
Figure 4 FSM of "Switch Avoidance Direction" 5
Figure 5 FSM of Hill Climb 6
Figure 6 Obstacle avoidance simulation 7
Figure 7 Inclination detection and redirection simulation 8
Figure 8 Obstacle avoidance on the slop simulation 8
Figure 9 Obstacle avoidance test map 9
Figure 10 Obstacle avoidance test illustration - set 1 9
Figure 11 Obstacle avoidance test illustration - set 2 9
Figure 12 Obstacle avoidance test illustration - set 3 10
Figure 13 Obstacle avoidance test illustration - set 4 10
Figure 14 Barrier wall avoidance test map 11
Figure 15 Barrier wall avoidance test illustration 11
Figure 16 Inclination detection and redirection test illustration - set 1 12
Figure 17 Inclination detection and redirection test illustration - set 2 12
Figure 18 Cliff detection and avoidance test 1 illustration 13
Figure 19 Cliff detection and avoidance test 2 illustration 13
Figure 20 Platform behaviour test illustration 14
Figure 21 Downhill redirection test illustration 14
Figure 22 Stop state test illustration 14
Figure 23 Final Finite State Machine 15
Figure 24 Final workshop map and actual operation route 16
Figure 25 Final run illustration - set 1 16
Figure 26 Final run illustration - set 2 16
Figure 27 Final run illustration - set 3 17