ELEN90066 - Embedded System Design
Final Project Report



Maxim Trentin: 1182823

Jarod Tang: 1175923

Xinyu Shen: 1171876

1. Introduction: 3
1.1 Aim and Objectives: 3
2. Finite State Machine: 3
3. Input Data: 4
3.1. Sensor Data 4
3.2. Other Inputs 4
4. Design Breakdown: 4
5. Testing Procedure: 5
5.1 Testing through CyberSim Simulator: 5
5.1.1 CyberSim Simulator Navigation Development 6
5.1.2 CyberSim Simulator Hill Climb Development 7
5.2 Limitations of CyberSim: 8
5.3 Testing in Person 10
6. Validation Procedure: 10
6.1 Downhill Descent: 10
6.2 Termination/Stopping Method: 11
6.3 Corner counter: 12
6.4 Final Statechart Logic: 12
7. Refinement Procedure: 13
7.1 Unique values: 13
7.1.1 Cliff Sensor: 13
7.1.2: Accelerometer range: 14
7.1.3: Distance traversed: 14
7.2 Averaging Function 15
8. Final Result: 15
8.1 Expected Result: 15
8.2 Actual Outcome: 16
9. Discussion: 17
10. Suggestions/Improvements: 18
11. Conclusion: 19
12. Bibliography 20
13. Appendices: 20
Appendix A: Equations 20
Appendix B: Extra Diagrams 21

1. Introduction:

In this report, we will discuss the work we conducted with the Kobuki robot and how the design was implemented such that the robot was able to navigate a random obstacle course. The foundation of the design was through the use of a Finite State Machine (FSM), which controlled the robot's behaviour and logic. The provided in-built sensors from the robot were also used in helping to identify obstacles and how to navigate them.

The Kobuki robot is comparable to an experimental mobile machine that can be used to implement classroom teachings and see how well the theoretical results compare with real-world results. The robot is equipped with binary pressure sensors, accelerometers and a gyroscope which are all useful features that can be used across a wide range of experiments. It is quite a popular machine for tests to be run on as it has a relatively low cost, is durable and is versatile.

For this final project, the binary bump sensors, docking IR sensor and accelerometer were the main sensors used. How they were each uniquely used will be discussed further and elaborated in the following subsections.

1.1 Aim and Objectives:

This project aims to develop an appropriate Kobuki algorithm in the form of a finite state machine such that it fulfils the obstacle course design requirements. Moreover, this project acts as a way to apply and iterate upon concepts taught by the Embedded Systems Design course.

2. Finite State Machine:

The brief logic of the FMS is entailed here, with a more in-depth description to come in later parts.

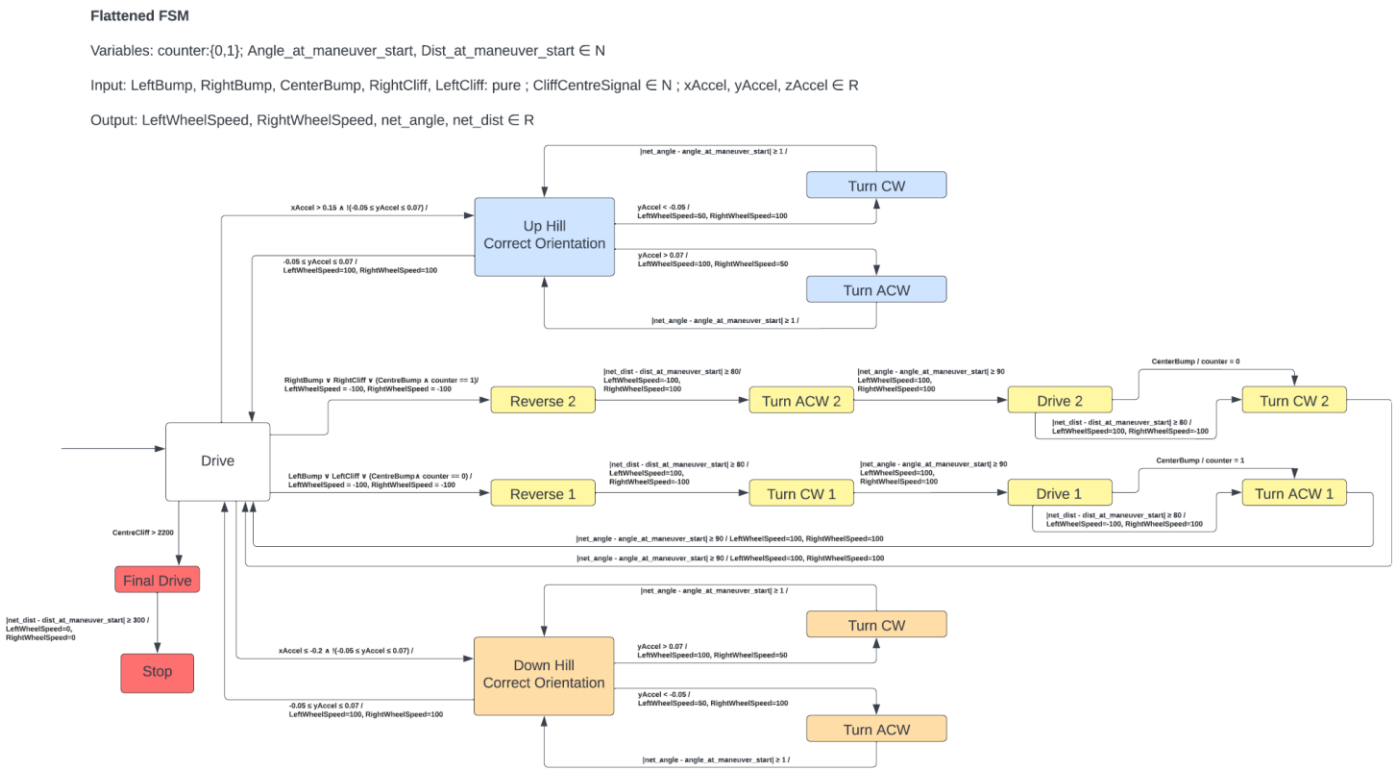


Figure 1: Kobuki Finite State Machine

3. Input Data:

Several sensors were utilised to help the robot complete the obstacle course and hill climb, including cliff sensors, bump sensors, and accelerometers. Counters and an averaging function are also implemented to achieve specific functions.

3.1. Sensor Data

Accelerometers

Accelerometers are used to constantly check the orientation of the robot on the ramp. The sensor reading of the vertical axis would be $cos(\theta)$, where θ is the ramp angle. When on flat ground, the vertical axis should always have sensor readings of around $1g$ ($9.81m/s^2$), as $\theta = 0$. When on the ramp, however, the vertical component of acceleration is reduced. This reduction in the sensor value can be used as an indication of whether the Kobuki is on flat ground or going up or down the ramp.

When the Kobuki is on the ramp, it will be tilted at an angle, ϕ (*Appendix B figure 26*). By analysing the Kobuki with rotation matrices (Appendix A Equation (1)-(4)), based on the y component of the rotation matrix expressed in equation (4), the readings will be $\sin(\theta)\sin(\phi)$, so if $\phi = 0$, meaning the Kobuki is perfectly aligned upwards, then the sensor readings will also be 0. Otherwise, the orientation of the Kobuki needs to be adjusted such that the y-axis reading is (as close as to) 0.

Bump Sensors

The Kobuki is equipped with 3 binary bump sensors, one in the centre and one on each side. These sensors were essential in the obstacle avoidance stage. Whenever a bump is activated, it triggers a process that makes the Kobuki turn to the side to avoid the obstacle. The processes will be explained in more detail in the design breakdown section.

Cliff sensors

There are 3 cliff sensors, one in the middle, and one on each side on the underside of the Kobuki. Cliff sensors provide analogue data of how far away the ground is from the Kobuki. When going on the ramp, and when getting off from the ramp, there will be a moment where the distance of the robot to the ground is not constant. When this is detected, processes would ensure that the Kobuki stays upright on the ramp and stops after a brief distance when leaving the ramp.

In addition, they also serve to keep the robot on the ramp and not fall off. When the Kobuki is near an edge, the cliff sensors on the appropriate side will detect a massive height change and veer the robot away from the edge.

3.2. Other Inputs

Averaging Function

The averaging function serves as a low-pass filter to stabilise some of the outputs. It can effectively reduce the noise received by the sensor readings for more reliable performance.

Counter

The counter is used to help make decisions in the obstacle avoidance stage. For instance, it is easy to tell the robot to go right when the left bump is triggered and vice versa. However, when the centre bump is triggered, it can be quite arbitrary as to which direction the Kobuki travels. The counter keeps track of how many collisions with the middle have happened, and changes the direction of the Kobuki if needed, for a more flexible obstacle avoidance algorithm.

4. Design Breakdown:

As seen in *figure 1*, there were three main categories of tasks that the Kobuki had to complete. The below image is a rough outline of the implementation of the algorithm, with the use of state refinements.

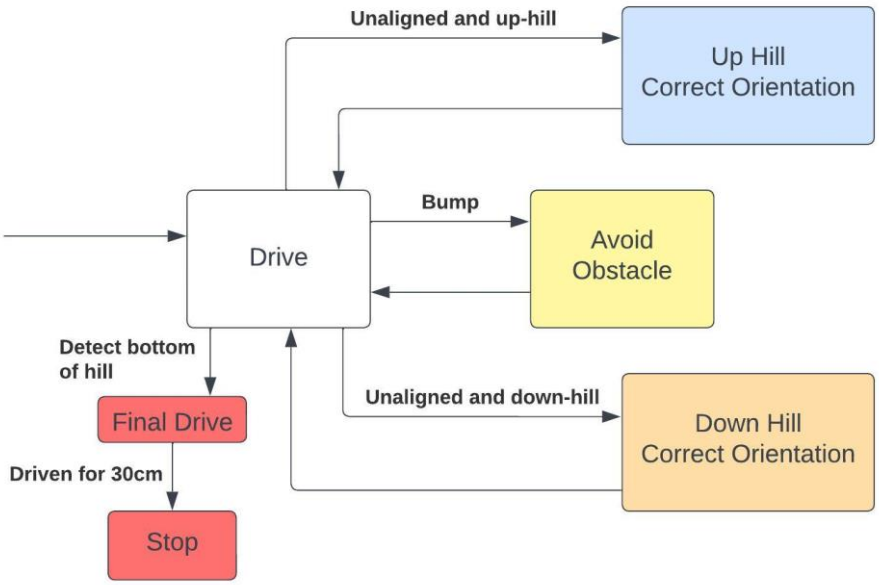


Figure 2: State Refinements

1. Obstacle avoidance: This section utilises bump sensors, as well as cliff sensors to navigate around the obstacles. If there is something in the way, this segment of code makes the Kobuki navigate in the other direction.

2. Hill Ascend orientation correction: By utilising the accelerometer data, whenever the x-axis has a reading higher than 0.15, the robot enters a hill ascend state. It also checks for orientation by reading the y-accelerometer. If the robot is heading straight up the hill, the y-accelerometer should be close to zero. The angle of the robot is adjusted degree by one degree until the y-axis acceleration is close enough to zero ($-0.05 \leq yAccel \leq 0.07$).

3. Hill Descend: This is identical to the hill ascend orientation correction, except this is for when the robot is going down-hill ($xAccel \leq -0.2$), the rotation is reversed compared to the up-hill section (CW rotation becomes ACW, and vice versa).

5. Testing Procedure:

5.1 Testing through CyberSim Simulator:

The testing of the Kobuki robot was conducted through the CyberSim Simulator Software in two main stages.

1. Obstacle Navigation: The first stage required the development of logic in a simulated environment that allowed the Kobuki robot to successfully navigate through a course filled with obstacles.

2. Hill Climb: The second stage was the development of an additional logic path that allowed the Kobuki robot to travel up a ramp directly.

These two stages were originally solved in workshops 4 & 5 respectively however, the logic was developed and implemented using C programming language. Although this meant that the logic developed in this phase couldn't be directly translated over into Statechart format, the basic exoskeleton had been laid out for us to follow when developing the Statechart.

There were some benefits though to us having exposure to how the C code implementation worked. By doing these earlier workshops, we were exposed and learned as to what types of data the Kobouki robot would be providing us to work with. As can be seen in *figure 3* below, the exact datatypes were shown to us in Workshops 4 & 5. This allowed us to understand how we could use these sensor readings whilst building our Statechart.

```
typedef struct{
  KobukiBumps_WheelDrops_t bumps_wheelDrops;
  bool cliffLeft;
  bool cliffCenter;
  bool cliffRight;
  uint16_t  cliffLeftSignal;
  uint16_t  cliffCenterSignal;
  uint16_t  cliffRightSignal;
  KobukiButtons_t buttons;
  uint16_t  LeftWheelEncoder;
  uint16_t  RightWheelEncoder;
  int16_t   LeftWheelCurrent;
  int16_t   RightWheelCurrent;
  int8_t    LeftWheelPWM;
  int8_t    RightWheelPWM;
  bool    LeftWheelOverCurrent;
  bool    RightWheelOverCurrent;
  uint16_t  TimeStamp;
  uint8_t   BatteryVoltage;
  chargerState_t chargingState;
  int16_t Angle;
  int16_t AngleRate;
} KobukiSensors_t;
#endif

typedef struct{
  bool  wheeldropLeft;
  bool  wheeldropRight;
  bool  bumpLeft;
  bool  bumpCenter;
  bool  bumpRight;
} KobukiBumps_WheelDrops_t;

/// Kobuki button sensors.
typedef struct{
  bool  B0;
  bool  B1;
  bool  B2;
} KobukiButtons_t;
```

Figure 3: C code variable types

A case in point: the *cliffCenter* data is a simple Boolean yes or no value whilst the *cliffCenterSignal* data is provided as a 2-byte unsigned integer. This information was key to how we developed certain chunks of our Statechart diagram which will be explained next.

5.1.1 CyberSim Simulator Navigation Development

For this Statechart logic, there are four main states, which can be seen in *Appendix B figure 21*, that were used to allow for the Kobuki to navigate through the obstacle course which can be seen below in *Figure 4*.

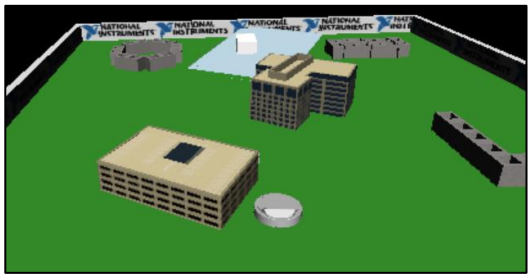


Figure 4: CyberSim Navigation course

1. **Drive State**: This is the default state that the robot will eventually aim to always revert to resulting in the forward drive of the robot towards the end goal. From this state, there are three possible transitions based on bump conditions:
    1. **RightBump**: When a true Boolean value is returned from the right bump sensor, this condition will be met, and the logic will transition into the *Reverse 2* state.
    2. **LeftBump**: When a true Boolean value is returned from the left bump sensor, this condition will be met, and the logic will transition into the *Reverse* state.
    3. **MidBump**: When a true Boolean value is returned from the centre bump sensor, this condition will be met, and the logic will transition into the *Reverse* state.
2. **Reverse States**: This state causes the robot to move backwards and away from whatever obstacle it just ran into. Both the *Reverse* and *Reverse 2* set the drive speed of the wheels to an equal negative value. Once a set distance of 100mm has been travelled, the logic progresses on to the next states.
3. **Turn States**: These states result in the robot turning either clockwise or anti-clockwise depending on which sensor was triggered by an obstacle being hit and what path was followed.
    - **Turn CW 1** & **Turn ACW1**: These states represent both clockwise and anti-clockwise turns of 90°. The sequence of these turns will be followed when the Kobuki robot runs into an obstacle and either the ‘bumpLeft’ or ‘bumpCenter’ sensors are triggered (see *Figure 3* for sensor value type). When this

path is followed, these changes in facing direction, $Straight \rightarrow Right \rightarrow Straight$, of the robot will result in the position of the robot shifting to the right in hopes to avoid the obstacle when it proceeds forward next.

- **Turn ACW2** & **Turn CW2**: These states represent both an anti-clockwise and clockwise turns of 90°. The sequence of these turns will be followed when the Kobuki robot runs into an obstacle and the 'bumpRight' sensor is triggered (see *Figure 3* for sensor value type). When this path is followed, the facing direction of the robot will change, $Straight \rightarrow Left \rightarrow Straight$, and result in the position of the robot shifting to the left in hopes to avoid the obstacle when it proceeds forward next.

4. **Temporary Drive States**: The *Drive temp 1* & *Drive temp 2* states represent the horizontal distance that will be travelled after the first turn states on either path have been passed. These states will result in the Kobuki robot driving a set distance in either the left or right direction in the hope of clearing the obstacle before attempting to proceed forward again.

A summary of how the Kobuki will navigate an obstacle depending on its location is provided in *Figure 5 below*.

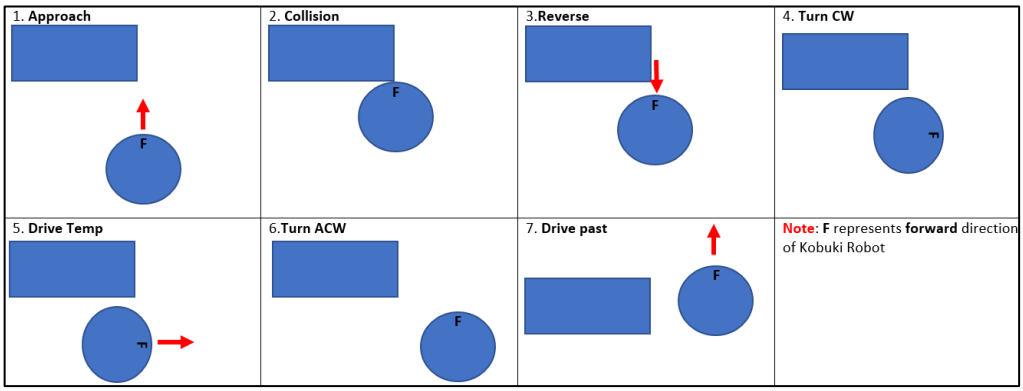


Figure 5: CyberSim Navigation Example

5.1.2 CyberSim Simulator Hill Climb Development

The Statechart logic for the Hill Climb simulation only involves 3 main states. *Appendix B Figure 22* illustrates how these states were organised in logic. *Figure 6* shows the hill that the Kobuki had to climb in the CyberSim simulation.

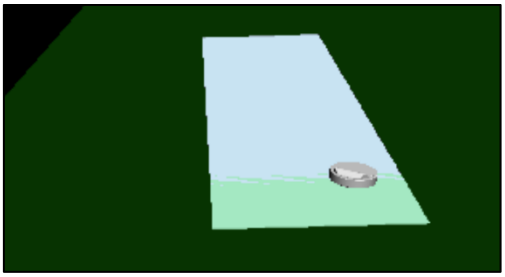


Figure 6: CyberSim Hill Climb Course

1. **Drive State**: This is the same default drive state as what is used in the navigation simulation as described in 5.1.1. The only difference is that there is just one condition that will the Statechart to leave this default state.
    1. **xAccel**: This condition looks at the magnitude of the acceleration in the x-axis. The x-axis is examined as its acceleration value should be near zero when the Kobuki rests flat on the ground or faces directly up a hill. If the x-axis acceleration is past a certain boundary condition, the robot will identify that it is on the hill but at an angle and needs to straighten. Refer to *Figure 7* & *Appendix B Figure 23* below to see the axis of the Kobuki in the CyberSim Simulation mode as well as the boundary conditions set in xAccel.

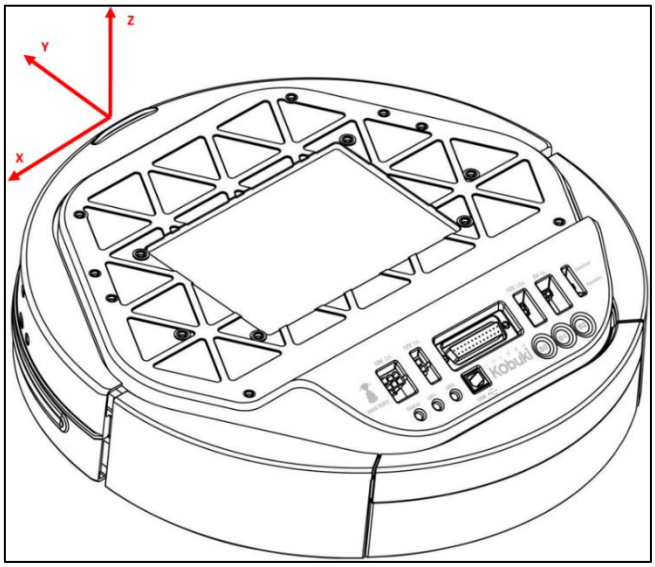


Figure 7: CyberSim Kobuki Axis

2. **Straighten On Hill**: This condition brings the Kobuki to a stop so that the orientation can be corrected before proceeding to drive. It can be viewed as an intermediary/decision-making state. This state was designed so that the Kobuki could safely adjust itself to make sure it was travelling upwards along the ramp without getting too close to the edges. The logic path that is followed after this state will be dependent on the value of the y-axis acceleration.
3. **Turn calibrate**: These two states will either rotate the Kobuki robot in either the clockwise or anti-clockwise direction in 2° increments. This small change in angle is to ensure that the boundary conditions on what classifies the robot as facing straight up the hill can be restricted to a tight range. This state may be visited many times until the Kobuki is within the range set by the *Straighten On Hill* state and is facing directly up the hill.

A summary of how the Kobuki will straighten itself on a hill is provided in *Figure 8* below.

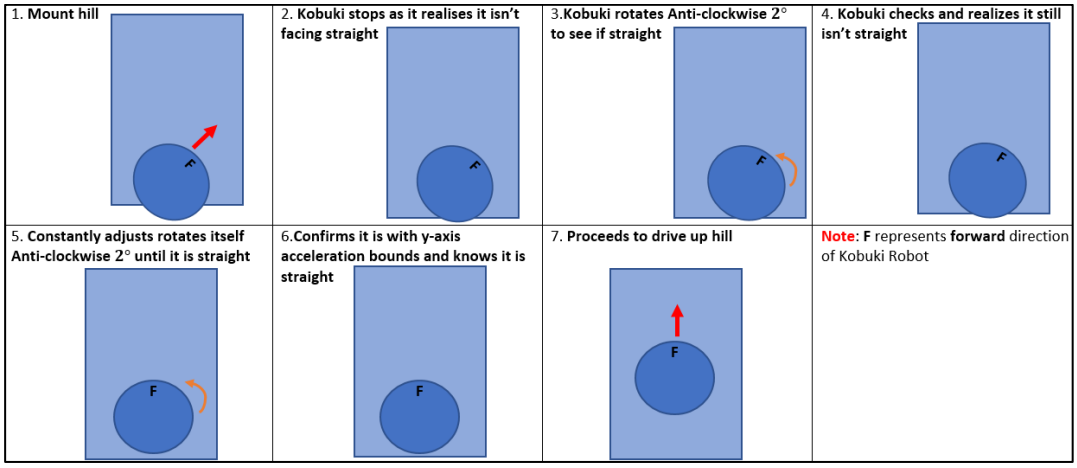


Figure 8: CyberSim Hill Climb Example

5.2 Limitations of CyberSim:

Whilst CyberSim is a very useful software as it helps us understand the obstacles and challenges the Kobuki robot must overcome, there are still some aspects of this project that are not covered by the software simulation.

Unrealistic performance:

The CyberSim software runs the Statechart logic in an ideal situation. However, it gives an unrealistic expectation as to how the real model will operate outside of the simulation. Some factors the simulation fails to consider are:

- Drift:
    - Although the Kobuki is supposed to drive straight forward, often the robots tend to drift in a certain direction. Over a period of time, this leads to the forward direction of the robot deviating from the original straight path it set out on.
- Rotational Inertia:
    - When turning on the spot, the simulation assumes that the robot will turn exactly 90° every time. However, due to rotational inertia, the robot can sometimes end up rotating more than 90° at times when the turn speed is set too high.
- Sensor read time:
    - In the simulation, as soon as the Kobuki hits an obstacle, it can immediately read from its sensors that there has been a collision. This in turn results in the Kobuki instantaneously stopping and performing its obstacle avoidance routine, see *Figure nav example*. However, the actual Kobuki does not have instantaneous processing. This means that the Kobuki is unable to drive too fast as it can sometimes lead to the robot 'climbing' up the obstacles due to the Statechart logic not reaching the stop and reverse instructions fast enough.

Incorrect Axis:

As shown earlier in *Figure 7*, the acceleration axis in the CyberSim software was defined with the Y-axis pointing in the forward direction. However, when the myRIO onboard sensor was attached, the X & Y axis was flipped as shown in *Figure 9*. This meant that all the axes in the logic for the simulation had to be changed in order for a Statechart to work with the physical robot in operation.

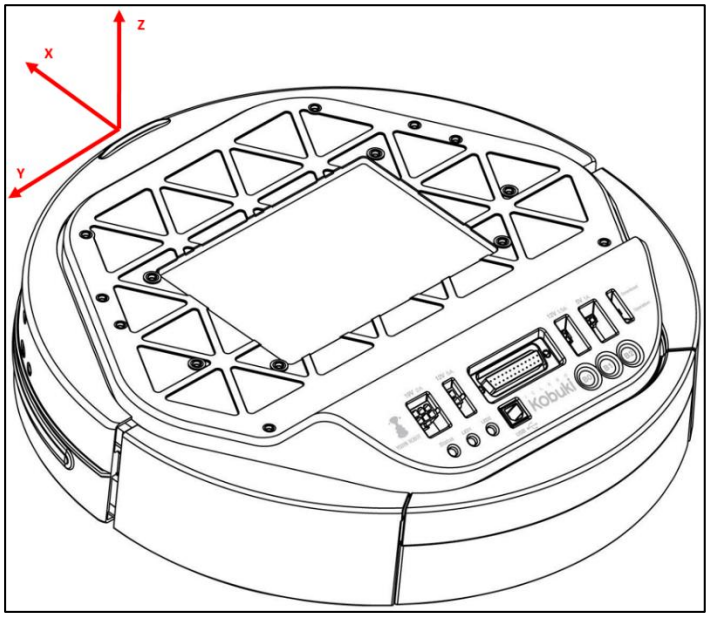


Figure 9: Real operating Kobuki Acceleration Axes

Sensor noise:

The most obvious shortcoming of CyberSim is that it fails to factor in sensor noise. A lot of the readings have nice numbers that round to 0 or give consistent, accurate readings whereas, in real world applications, there is a lot of variation that can occur. Some examples of these variations/noise are:

- Offset Accelerometer Readings

- On flat ground, the X or Y-axis accelerometer readings can sometimes be offset to a value that is not zero. This means that in the hill climb segment, adjustments for these offsets must be considered and made.

- Cliff Sensor Readings
  - The infrared reading in the CyberSim model always returns a constant value back for the cliff sensor readings on flat ground. However, this reading varies when using the robots and fluctuates a lot between each individual robot too. This means that for each new robot we ended up working with, we had to adjust our cliff sensor values.

5.3 Testing in person

After completing and passing the CyberSim tests, we compiled both the Navigation and Hill Climb logic into one encompassing Statechart. This was a smooth transition as both state charts originally shared the same default drive state. This meant we could keep this as the common drive state where conditions would leave from and eventually return to. The combination of these two state charts is shown in *Figure 10* below.

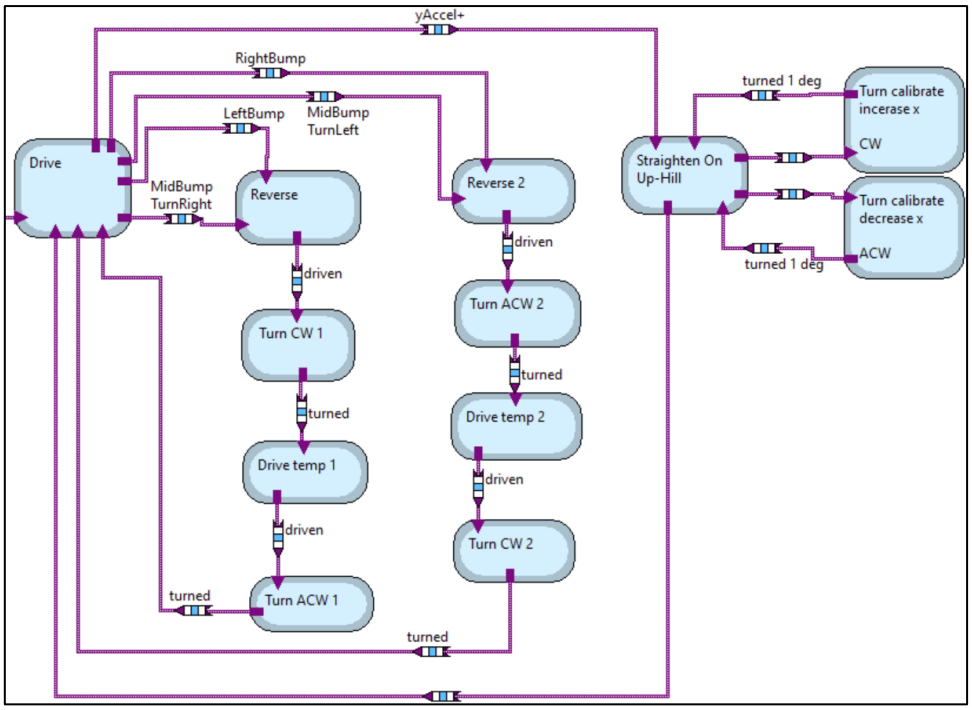


Figure 10: Navigation and Hill Climb Combined Statechart

Whilst this Statechart did allow for the Kobuki to successfully navigate the obstacle course, it fell short of completing all the goals of this project. There were three main areas from here that needed to be developed/worked on: the Downhill Descent, the Termination/Stopping method and Corner Recognition. These 3 areas were developed to make the Kobuki Statechart logic valid, and able to complete the entire course. The development of these areas will be discussed in *Section 6*.

6. Validation Procedure:

6.1 Downhill Descent:

With the Statechart implementation shown in *Figure 10* above, the Kobuki is able to reorientate itself so that it is facing straight when it is traversing uphill. However, when descending a hill, this Statechart is inadequate as it will simply reorientate the Kobuki to face straight uphill again. This is due to the opposite signs that will be in front of the acceleration value for the y-axis depending on whether the Kobuki is on the uphill or downhill slope. *Figure 11* below illustrates how the y-axis value on the uphill segment for a scenario that requires an anti-clockwise adjustment will

have a positive value. However, a Kobuki on the downhill segment also in need of an anti-clockwise adjustment will have a negative y-axis acceleration value.

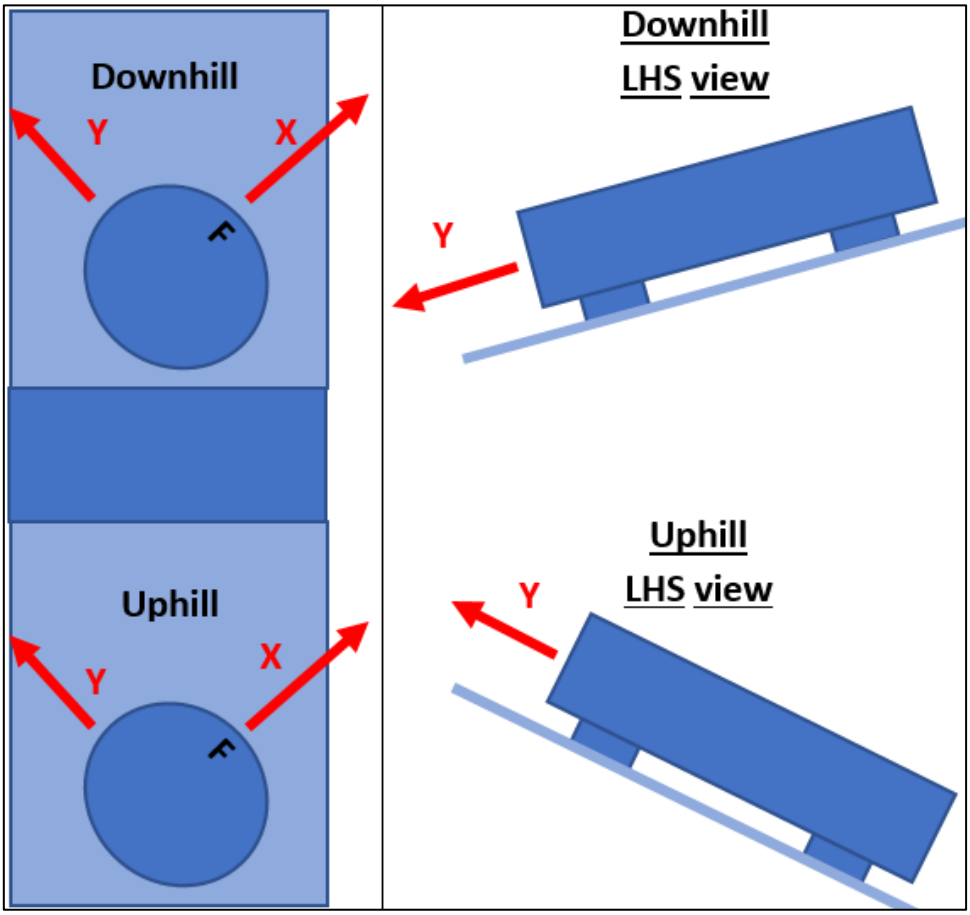


Figure 11: Uphill v Downhill acceleration reading

In order to fix this problem, a simple addition of another rotation method like what is shown in *Figure 22* is required. The only difference for these new states is that the condition of the guards will be the opposite of the original uphill guard conditions. The last adjustment that will be made as well is the use of the accelerometer in the x-axis to control the flow of logic and determine whether the Kobuki is on the uphill or downhill slope. By including this in the guard condition that leaves the default drive state, we will be able to control which *Straighten On Hill* method is followed.

6.2 Termination/Stopping Method:

At the end of the run, the Kobuki must stop and terminate within a small distance of the end of the downhill ramp. This can be achieved by analysing the cliffCentreSignal shown in *Figure 3*. This value is a 2-byte unsigned integer and represents the intensity of an infrared signal relaying between the front of the Kobuki and the ground. At the start and end point of a ramp, this value increases as the front of the Kobuki gets closer to the ramp/ground before the wheels also reach the ramp and start to lift and change the angle of the Kobuki. *Figure 12* below visually demonstrates this concept.

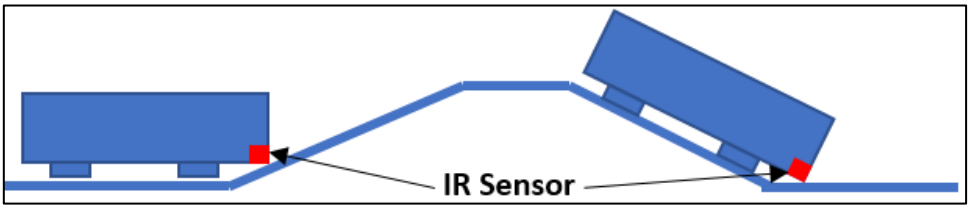


Figure 12: Increasing IR Sensor Value Readings

Using this fact now, we can find what value the IR sensor reading jumps to at these stages so we can use it in identifying the bottom of the downhill ramp. Once this value has been found, we can use this along with the accelerometer on the x-axis to accurately determine when the Kobuki is at the end of the course. Once this guard has been met, it is just a simple case of driving a small set distance before commanding the Kobuki to terminate.

6.3 Corner counter:

The last area that is of concern has to do with how intelligent our Kobuki is. Previously, when adjusting either left or right due to an obstacle collision with the *bumpCenter*, the Kobuki was required to drive a set distance before turning back straight as illustrated in the *Figure nav example*. However, this logic will not hold up in the unique case when the Kobuki finds itself in a corner. If the robot were to end up in a corner with its current Statechart logic, the Kobuki would keep mindlessly driving into the same side wall.

To mitigate this issue, a corner counter mechanism was introduced. This counter starts off with a default value of 0 and changes only when the Kobuki is cornered and may only ever toggle between the values 0 and 1. When the counter is at 0, the Kobuki will turn to the right upon detecting an obstacle from the *bumpCenter* sensor. However, when the counter equals 1, the Kobuki will turn to the left. An extra guard condition has been added to the navigation logic, where the counter's value only changes if the robot comes across another barrier whilst it is still in either one of its *drive temp* states. The counter will either increase or decrease based on the direction of movement. Consequently, the robot enters the next state as it bypasses the previously specified distance criterion through this new additional guard. With this new counter value, the direction that the *bumpCenter* sensor would cause the Kobuki to rotate would change. *Figure 13* below illustrates an example of this logic.

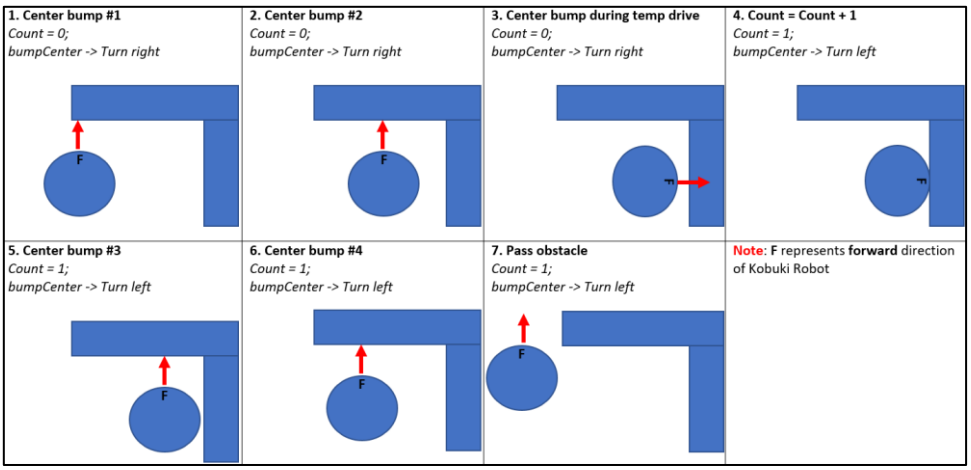


Figure 13: Corner Counterexample

6.4 Final Statechart Logic:

With the 3 key errors identified and solved in segments 6.1, 6.2 & 6.3, we can compile them all together into the final form of our Statechart. This final Statechart can be seen in *Figure 14* below and shows the updated, navigation logic with the new count guards, the updated hill climb logic that can sense when the Kobuki is on an uphill or downhill slope as well as the final exit strategy.

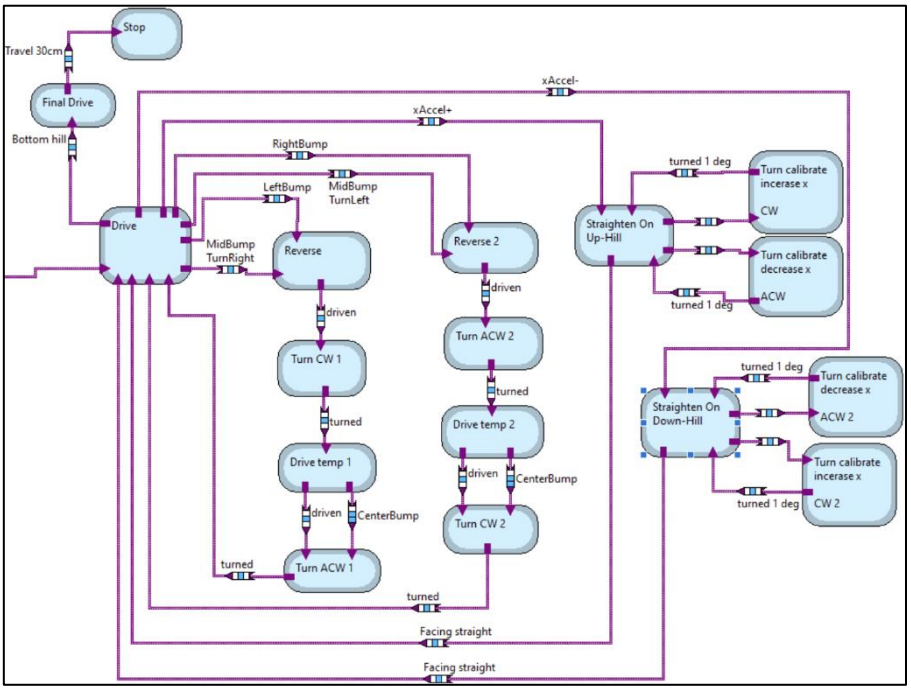


Figure 14: Final Statechart

This Statechart is what we refer to as our "safe" logical state chart. It only performs very plain and simple movements whilst also being intelligent enough to understand its location on the obstacle course, unlike previous versions like what can be seen in *Figure 10*.

7. Refinement Procedure:

As mentioned earlier in *section 5.2*, the modelling of the device can work well in theory however, not as well in practice. This can be due to imperfect sensor values being used instead of the ideal perfect values which are provided in simulation and theoretical environments. These imperfections lead to the need for unique values for certain sensors to be established so that each individual robot can be compatible with our final Statechart logic.

7.1 Unique values:

7.1.1 Cliff Sensor:

The 2-byte unsigned integer IR sensor reading that is provided by the *cliffCenterSignal* typically is around 2000 when placed flat on the ground. This normally gives us a good gauge as to what value we can use in the guard for trying to determine when the exit sequence should commence, typically a value above 2300 sufficed. *Figure 15* below shows that the final Kobuki we used on the presentation/assessment day had a reading of around 1940 on a flat surface.



Figure 15: Unique values for Kobuki Robot

However, there were some workshops where the IR sensor was returning a value of around 1600 on flat ground. This meant that the condition for the exit sequence had to be uniquely toggled to an appropriate value based on which Kobuki robot we ended up using on the day.

7.1.2: Accelerometer range:

The accelerometers theoretically should display 0 when the Kobuki is stationary on a flat surface. However, as can also be seen in *Figure unique values for Kobuki* above, this was not the case and there were unique offset values to each axis. This meant that the range we had set to determine whether the robot was flat or facing straight up the hill had to be adjusted by an appropriate factor. For the final assessment, it can be seen that our boundaries on the x-axis had to be reduced by 0.04 whilst the boundaries on the y-axis had to be raised by 0.02. This then allowed for our logic to be followed as expected by the Kobuki due to the calibration now being complete.

7.1.3: Distance traversed:

The distance that was traversed horizontally when the Kobuki was attempting to navigate an obstacle was never flagged as a critical error due to logical values being chosen. However, there is a chance that too small or too large a value could be chosen making the application of our Statechart logic impractical.

If too small of a distance is traversed, lots of time will be wasted each time the Kobuki runs into an obstacle, as it will take many attempts for enough distance to be covered to allow the robot to pass by. On the other hand, if too large of a distance was chosen, then the robot had the potential to pass the gap between objects. This would lead to the Kobuki assuming a solid wall was placed in front of it and it would be unable to find an appropriate gap to pass through. Knowing these scenarios, the horizontal distance traverse in the drive state was set to 150mm which is roughly ½ the length of the longest obstacle. This meant that at worst, the Kobuki should only ever run into an obstacle 3 times at most (provided it isn't in a corner) before finding a path around it. This resulted in the Kobuki taking a safe but relatively efficient route when navigating the obstacle course.

7.2 Averaging Function

When ascending and descending the ramp, the Kobuki would sometimes stutter when adjusting to face straight up the hill. This was due to the y-axis accelerometer reading drastically fluctuating values due to the fast rate the wheels would spin when turning. When this occurred, the Kobuki would end up changing direction on the spot between clockwise & anti-clockwise rotations as the accelerometer readings would jump between negative and positive values. To fix this issue, an averaging function was introduced when reading acceleration data from the y-axis.

As per its namesake, the averaging function would take the average of the last N accelerometer readings along the y-axis and check if the average acceleration along the y-axis ($a_{y\,avg}$) was within the range $-0.05 \leq a_{y\,avg} \leq 0.07$. If the average value was outside this defined range, the Kobuki would take corrective action by entering an appropriate turning state. The specific corrective actions taken are all depicted in *figures 24 and 25* in *Appendix B*. The value of was initially set to 9, however, after observing that the Kobuki would take too long to straighten itself when it met the incline of the hill, it was reduced to a value of 3. This change did not affect the performance of the hill-climb algorithm.

8. Final Result:

8.1 Expected Result:

Based on the results within CyberSim and the success of workshop 6, we expected our finite state machine (FSM) to work and achieve the following obstacle course design requirements:

1. **Ground Orientation**: Ground orientation is the direction the robot faces when it first runs. It should not change after subsequent pausing/resuming interruptions. It will only be reset either due to a power cycle, reprogramming or restart of the robot or myRIO.
2. **Drive**: The robot will maintain ground orientation whilst driving and satisfy the following characteristics:
    a. **Obstacle Avoidance**:
        i. **Cliff Avoidance**: The robot will not fall off cliffs or edges.
        ii. **Wheel Hazard Avoidance**: The robot will avoid one or more wheels losing contact with the ground. If one or more wheels lose contact with the ground, the robot will recover.
        iii. **Object Avoidance**: The robot will avoid objects in its path. It can make contact with the object, so long as the robot immediately changes course in an attempt to avoid it. Moreover, this must always be satisfied, even if multiple objects are encountered simultaneously or in short succession.
        iv. **Reorientation**: After avoiding an object, the robot will reorientate to ground orientation.
    b. **Hill Climb**: The robot can climb hills.
        i. **Hill Climb**: If an incline is encountered, the robot will drive straight up the incline whilst avoiding cliffs.
        ii. **Hill Plateau**: The robot will continue driving along the plateau once the top of the incline is reached.
        iii. **Hill Descend**: The robot will be able to drive straight down a descent whilst avoiding cliffs.
        iv. **Ground Stop**: The robot will stop and terminate execution within a set distance of 40cm after climbing and descending.
3. **Performance**: The robot must satisfy the following performance characteristics:
    a. **Turnabout**: The robot will never rotate in place more than $180°$.
    b. **Chattering**: The robot will not move erratically or exhibit chattering.
    c. **Abnormal Termination**: The robot will not terminate for an arbitrary reason. Power or mechanical failures are exceptions to this.
    d. **Timeliness**: The robot will achieve its goals in a timely manner (within 180 seconds).

8.2 Actual Outcome:

In the final assessment, the final obstacle course was laid out as shown in *figure 17*. Each group was given four attempts in total, with the remaining attempts being used to get the fastest obstacle course completion time. Our Kobuki was not able to complete the obstacle course during the first, second and fourth attempts, however, we were successful on our third attempt.

On the first and second attempts, the Kobuki would repeatedly get stuck at the point marked by the red circle in *figure 17*. The following events that caused the Kobuki to get stuck are depicted in *figure 16*. These sets of events would repeat themselves 2-3 times before an intervention. The most probable reason for this scenario occurring is due to the drift shifting the ground orientation rightward. When the robot re-orientated itself back to its ground orientation, due to the drift, the robot overturned relative to its true ground orientation. Therefore, when the robot collided with the object shown in event 3, the robot thought that there was an object on its left and manoeuvred right to avoid it. To address this for the third attempt, changes to the wheel speeds were made. More specifically, the right wheel speed was made faster than the left. This is elaborated upon further in *section 9.* In addition, the pre-set distance that the Kobuki would travel in its object avoidance state was increased.

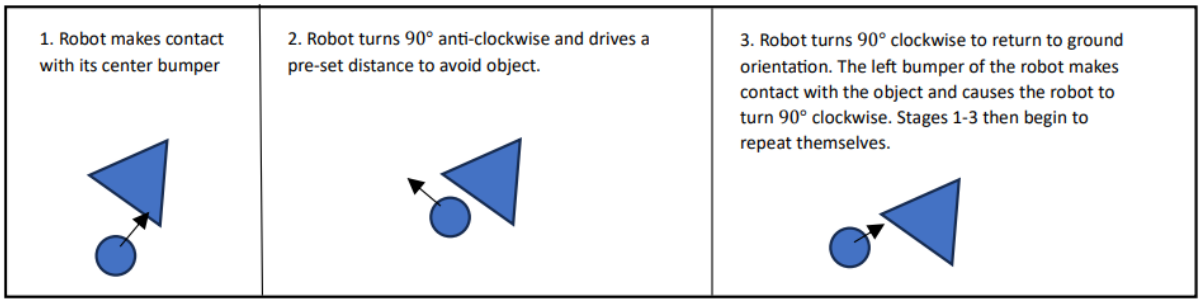


Figure 16: Scenario where the Kobuki got stuck

During the speed run attempt (attempt 4), the wheel speed of both the right and left wheels was increased to the maximum possible setting. Changing this setting seemed to mitigate the effect of the drift. However, the Kobuki would still find itself in the scenario depicted in figure *16*.

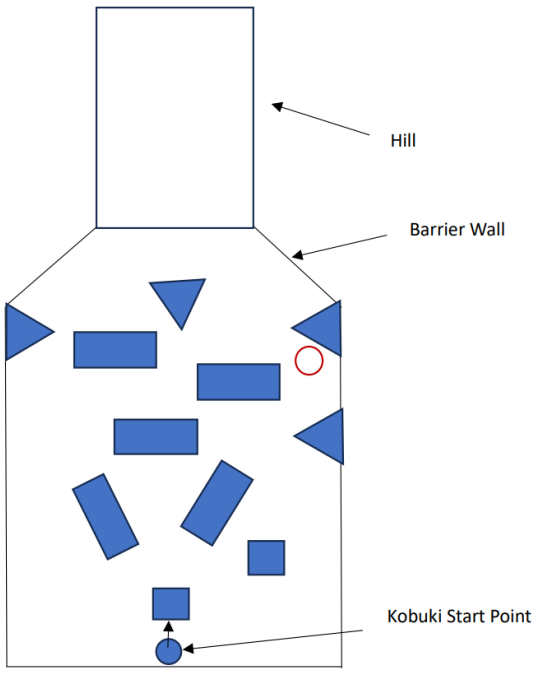


Figure 17: Final obstacle course layout. Note that items are not to scale.

Overall, the Kobuki struggled the most during the navigation section of the course. Once the Kobuki reached the hill climb section, it would consistently perform to the obstacle course design requirements and complete the hill climb section with no problems.

9. Discussion:

The major problem encountered was that the Kobuki was subject to large amounts of drift, being that it tended to drift rightwards whilst in drive mode. This was particularly noticeable when the Kobuki was in drive mode for long periods of time, as the robot would make substantial deviations from its ground orientation, impacting its ability to maintain its initial heading. Prior to the final assessment, this behaviour was not observed, so potentially this behaviour was due to mechanical issues with our specific Kobuki. Such issues could include slight motor differences or wheel friction variability.

Moreover, to counteract this, the right wheel was set to spin faster than the left wheel. We would determine the right wheel speed by trial and error, where we would let the Kobuki travel along its ground orientation over two metres. We would keep repeating this test until the Kobuki could travel in a straight line over two metres for a given right wheel speed value. After rigorous testing, the value of the right wheel speed was set to 201mm/s. Whilst this did help to mitigate large amounts of rightward movement, small amounts of drift were still visible. Moreover, the amount that the wheel speed is changed would be different for each Kobuki, making it so that testing would have to be done for each Kobuki if this particular solution were to be implemented.

In preparation for the speed run attempt (attempt 4), our group increased the wheel speed to the maximum possible setting of 250mm/s. It was observed that increasing the wheel speed helped to mitigate the effects of drift. However, when the Kobuki entered its object avoidance state and turned at such speeds, the front of the Kobuki would lift off and sometimes cause the Kobuki to terminate execution. Moreover, this would sometimes occur when the Kobuki collided with objects at such speeds. Therefore, this limited the maximum speed we could set the Kobuki to.

As stated in *section 8.2*, our hill-climb algorithm performed exceptionally well in the final assessment. Both the reliability and robustness of our algorithm can be attributed to the averaging function implemented, as this prevented many of the issues observed, such as chattering. Curiously, the effects of drift did not seem as apparent during the hill climb and descend segments of the course. This is mostly likely due to the low pass filter mitigating the impacts of sensor error and external disturbances. However, the smooth surface of the hill could also have been a more favourable environment for the Kobuki.

Throughout this project, much of the taught content from Embedded System Design was used to inform the design of algorithm, specifically content pertaining to sensors and actuators, hierarchical state machines, proportional control and reachability analysis was incredibly helpful. Hierarchical state machines formed the basis from which we developed our finite state machines from. The knowledge regarding creating separate state machines for different tasks and combining them was especially useful, given that our group chose to further the development of the Kobuki algorithm using Labview. In the latter stages of development and optimisation, reachability analysis helped us ensure the behaviour of the system and detect potential issues such as deadlocks or unwanted states. Whilst it was too difficult to derive a flat composed FSM and apply Breadth First Search (BFS) or Depth First Search (DFS), model abstraction was used to logically deduce how each case would be handed. Sensors and actuators helped us realise the failbacks that real-life sensors have and forced the consideration of estimation and mitigation techniques. The content pertaining to proportional control was considered in our potential improvements and explored further in *section 10*.

10. Suggestions/Improvements:

In reflecting upon the algorithm implemented for the final assessment, we recognised that the drift mitigation element of the algorithm could be greatly improved. A more elaborate guard could be implemented, which would continually check that the current heading of the Kobuki is always within an acceptable range, close to its ground orientation. This solution would have been more robust and easier to integrate into different Kobuki's. *Figure 18* is the guard written in C, where theta represents an acceptable range in degrees.

```
else if (state == DRIVE) {
    angleAtManeuverStart = netAngle;
    distanceAtManeuverStart = netDistance;
    if (abs(netAngle) <= theta) {
        state = DRIVE;
    }
    else {
        if (netAngle < -theta) {
            state = TURN_RIGHT;
        }
        else if (netAngle > theta) {
            state = TURN_LEFT;
        }
    }
}

else if (state == TURN_LEFT || state == TURN_RIGHT) {
        angleAtManeuverStart = netAngle;
        distanceAtManeuverStart = netDistance;
        if (abs(netAngle) <= theta) {
            state == DRIVE;
        }
    }
```

Figure 18: Drift countermeasure guard

Alternatively, this could also be achieved through proportional feedback control, where the current heading is constantly checked against the desired heading (ground orientation). This system would mitigate the error caused by drift by inputting a control action that is proportional to the error signal $E(s)$. The error signal is the difference between the desired and current headings. Further research and study are required to derive an appropriate model of the system, as well as suitable gain value to ensure stability. A potential control system is depicted in *figure 19*, where K_P represents the controller and $G(s)$ is the plant. The current and desired headings are represented by $C(s)$ and $R(s)$ respectively.

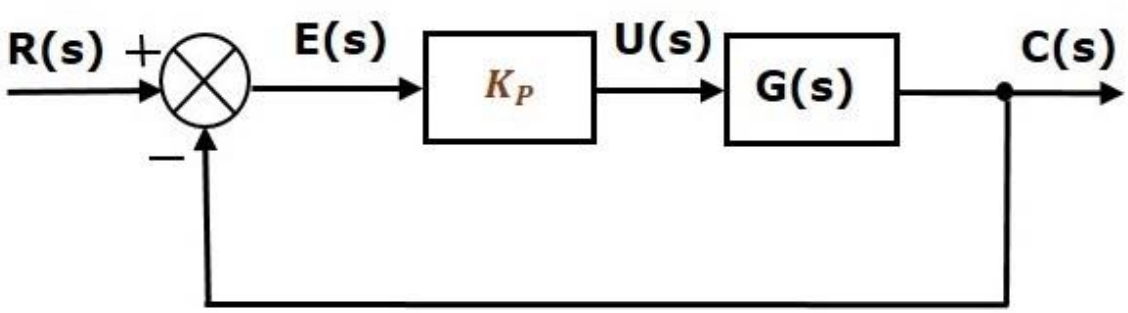


Figure 19: Proportional Feedback Control (tutorialspoint, n.d.)

In the interest of decreasing the course completion time, the reverse functionality could be removed altogether. Naturally, time would be saved when the robot enters its object avoidance state as it now would only turn either right or left, instead of first reversing.

Furthermore, an ultrasonic sensor system could be implemented. An ultrasonic sensor could detect objects before a collision occurs, enabling the robot to manoeuvre around the object pre-emptively. Moreover, this could help to

decrease the course completion time, as a more direct and shorter path could be taken. However, this would require further research into both the physical and software integration of such a system.

Lastly, although the edge case depicted in *figure 20* was not encountered in the final assessment, if the Kobuki algorithm was to be improved upon in the future, this would be a scenario worth considering.

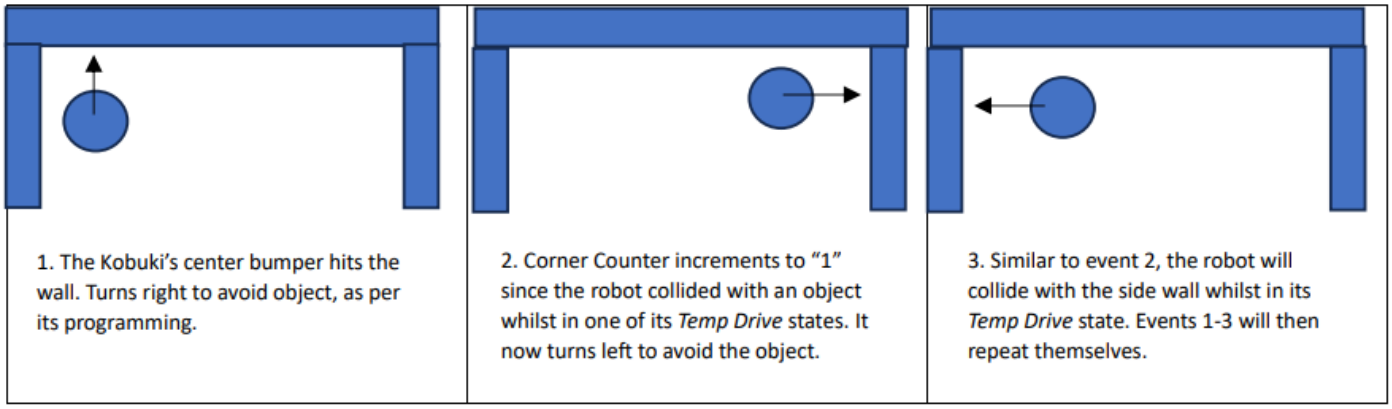


Figure 20: Edge Case

The issue arises as the Kobuki will always try to go in the direction of its ground orientation. However, it does not realise that the object it is trying to avoid encloses it. To avoid this problem, the Kobuki could be equipped with a mapping system such as SLAM (Simultaneous Localisation and Mapping). This would help the robot understand its surroundings better and avoid areas it has either recently visited or collided with. Alternatively, another global variable could track the number of consecutive collisions, such that after a certain number of consecutive collisions, the robot could take an alternative path in an attempt to break the loop.

11. Conclusion:

Over the course of this project, we successfully refined and developed a finite state machine (FSM) for the Kobuki robot to fulfil the obstacle course design requirements (see *section 8.1* for details). Materials from the lectures and workshops were instrumental in helping us establish good design procedures, enabling us to iteratively refine and improve upon our implemented Kobuki algorithm. More specifically, we found content pertaining to sensors and actuators, hierarchical state machines, proportional control and reachability analysis particularly useful.

Having access to both simulated and real-world environments was immensely helpful in the development of our algorithm. The simulation, whilst primitive, enabled us to verify the correctness of the logic of our algorithm. Real-world testing allowed us to uncover physical failbacks and redesign our algorithm such that it would be robust enough to deal with such conditions.

As stated in *section 9*, our hill-climbing algorithm was incredibly successful, demonstrating itself to be robust and without issues. Moreover, our issues were mainly present in the navigation section of the course, where during the final presentation, the Kobuki would repeatedly find itself stuck in the scenario depicted in *figure 16*.

From this experience, several potential improvements have been discussed (see *section 10* for details), with the improvements mainly aiming to improve the robustness of the algorithm, as well as its performance against the obstacle course design requirements. Overall, through the application of the theoretical concepts taught in this subject, this project helped us develop an understanding and appreciation of embedded systems, as well as a grasp of the many challenges faced by modern embedded engineers.

12. Bibliography

1. tutorialspoint. (n.d.). *Control Systems - Controllers - Tutorialspoint*. Www.tutorialspoint.com. https://www.tutorialspoint.com/control_systems/control_systems_controllers.htm

13. Appendices:

Appendix A: Equations

$$ {}^{0}_{1}R = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, {}^{1}_{2}R = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1) $$

$$ {}^{0}_{2}R = {}^{0}_{1}R\,{}^{1}_{2}R = \begin{bmatrix} \cos(\theta)\cos(\phi) & \cos(\theta)\sin(\phi) & -\sin(\theta) \\ -\sin(\phi) & \cos(\phi) & 0 \\ \sin(\theta)\cos(\phi) & \sin(\theta)\sin(\phi) & \cos(\theta) \end{bmatrix} \quad (2) $$

$$ ({}^{0}_{2}R)^{T} = {}^{2}_{0}R = \begin{bmatrix} \cos(\theta)\cos(\phi) & -\sin(\phi) & \sin(\theta)\cos(\phi) \\ \cos(\theta)\sin(\phi) & \cos(\phi) & \sin(\theta)\sin(\phi) \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (3) $$

$$ {}^{2}_{0}R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \sin(\theta)\cos(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\theta) \end{bmatrix} \quad (4) $$

Appendix B: Extra Diagrams

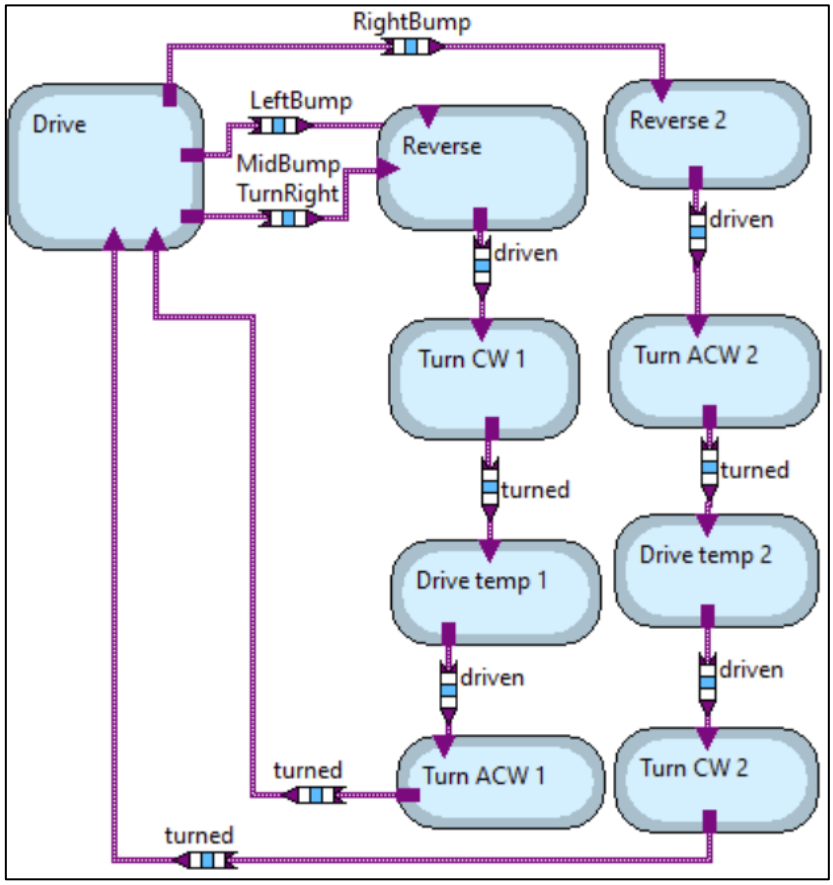


Figure 21: CyberSim Navigation Statechart Logic 21

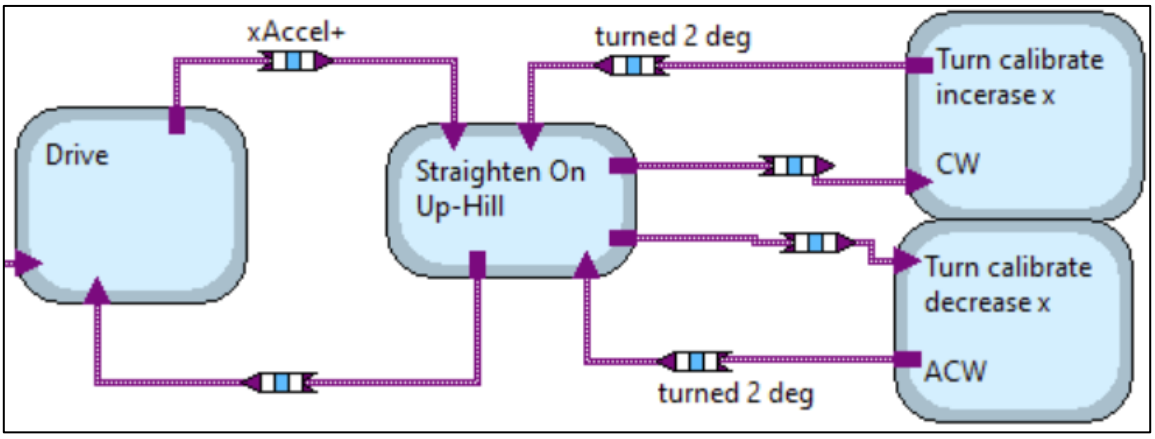


Figure 22: CyberSim Hill Climb Statechart logic 22

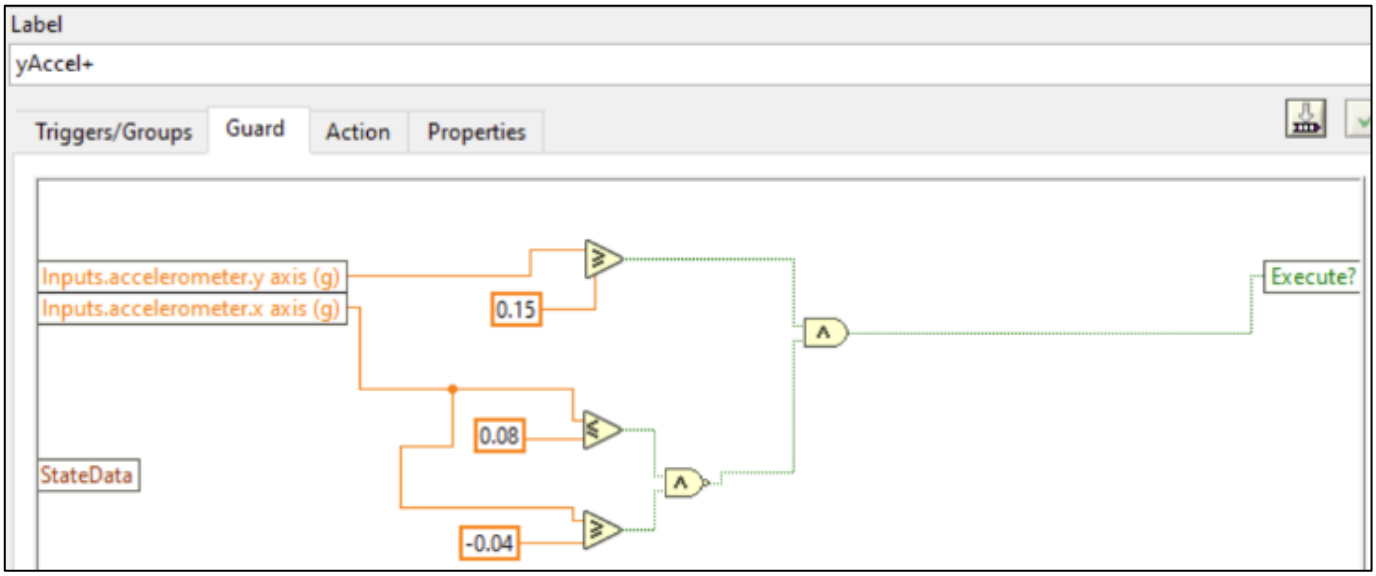


Figure 23: xAccel boundary conditions 23

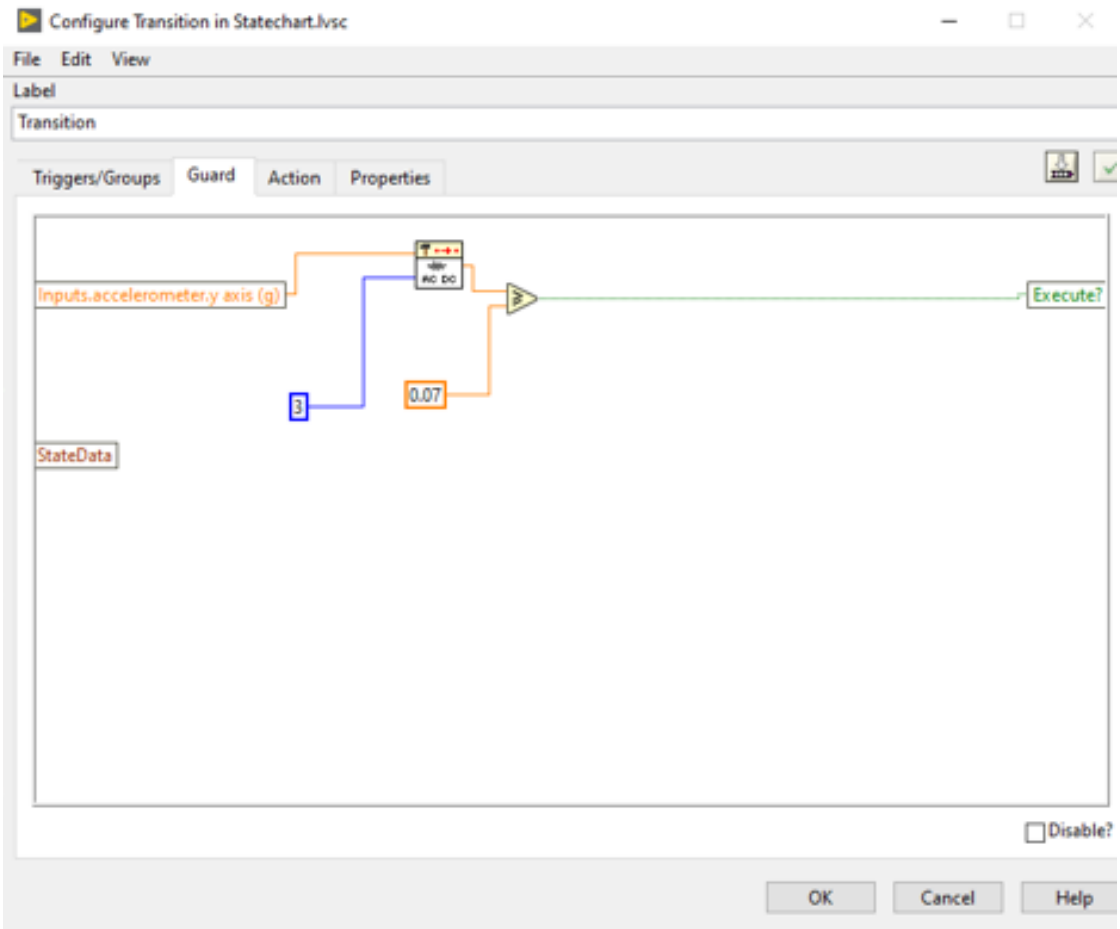


Figure 24: Averaging Function Upper Acceleration Bound

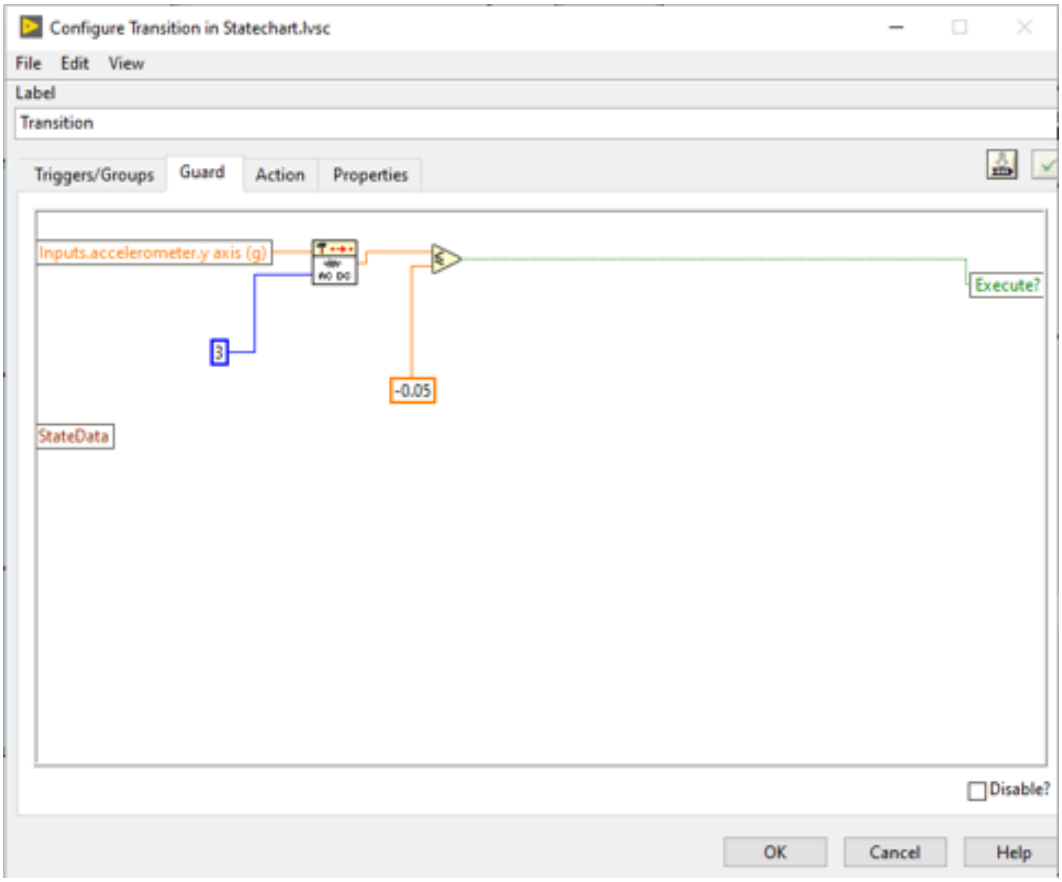


Figure 25: Averaging Function Lower Acceleration Bound

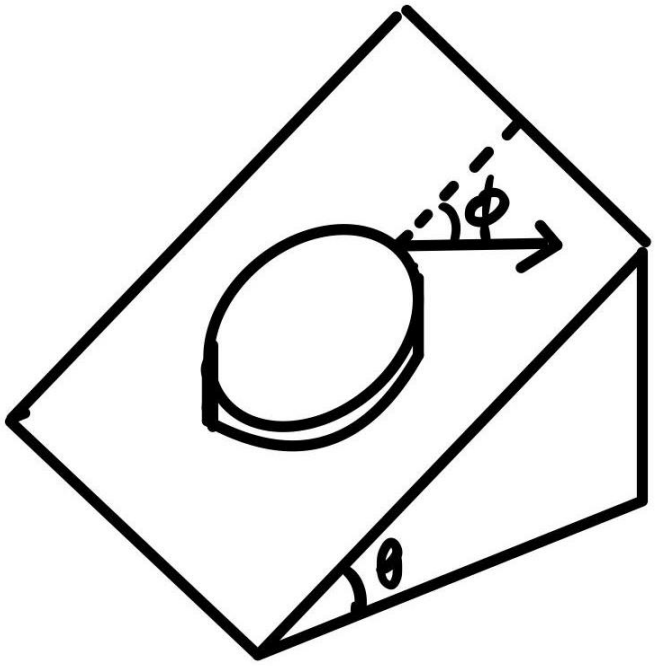


Figure 26: Isometric drawing of Kobuki on the hill