Edge Detection Robot Simulation Using Real Time DEVS

Ritvik Joshi
Carleton University
Department of Systems and Computer Engineering
Ottawa, ON
ritvik.joshi@cmail.carleton.ca

ABSTRACT

The use of robots for the automation of risky tasks is increasing day by day. Robots are more useful for the tasks that humans cannot achieve. These tasks need to be simulated for observing their behavior and performing optimization. DEVS is one of the well-established formal modeling techniques that can be applied to real time systems. In this paper, I am showing a simulation model for the edge detection model using Real – Time Discrete Event System Specification. Real Time DEVS formalism provides system development, which is modular and hierarchical, which helps in modeling systems unambiguously. The model consists of different types of components for the systems such as motor, sensor, and logic for motion control. The system is sub divided into atomic models and coupled models. All the modules are implemented using Cadmium simulation environment. The effectiveness of the developed model is manifested by simulating the model in various scenarios by flashing it on hardware. The model is simulated within the PC using simulation framework and simulated on actual robot shield. The simulation provides effective tools for research in the field of robotics. The modularity of DEVS makes it easier to modify and extend the model for further use. In summary, the edge detection robot simulation model provided in this study exhibits the strength and flexibility of the RT-DEVS formalism for modelling and simulating complex systems. The model provides a useful tool for design and analysis of edge detection robots and provides a base for which can be easily extended.

Keywords

Modeling, Simulation, real time systems, robotics.

1. PROBLEM AND MOTIVATION

The use of robots in production, defense, and aerospace is increasing day by day. Robots give easy ways to perform various tasks which involve hazard. The advancements in the technology for sensors backs the capabilities of performing complex tasks with higher precision. One of the crucial tasks in the field of robotics is edge detection. This is widely used in multiple applications such as navigation, object detection, and surveillance.

Developemt of reliable and efficient techniques for robots is challenging as it has time criticality associated with the task. The simulation of technique gives us the ability to observe the behavior and performance of robots in various scenarios. This also gives us an opportunity to optimize the system performance and improve the system reliability. The PC simulation results should give a robust solution, but the solution should also be valid and practical in the real world. The solution obtained in the result must be tested in real world environment. The Cadmium framework [5] for simulation gives the flexibility of developing a simulation which is valid in the simulation model for a PC based simulation but at the same time use the same base and flash the software in hardware and check the performance of the same model real world.

In this paper I am developing a model which simulates the behavior of an edge detection robot. The methodology for development, tools, contributions, and results are discussed further in the paper. The main goal of this project is to develop a simulation model using RT-Cadmium [2] for an edge detection robot that can accurately detect edges in various scenarios. The model should be modular and hierarchical, allowing for easy modification and extension. Additionally, the model should be efficient and scalable, allowing for large-scale simulations.

2. BACKGROUND AND RELATED WORK

In this section, we provide a brief overview of related work.

DEVS Formalism

DEVS (Discrete event system Specification) [1] is a formal framework for modeling and simulation of discrete event systems. The framework is based on hierarchical structure of atomic components. The modeling and simulation are separated in DEVS for increasing the model reusability and interoperability. The atomic model is the most fundamental part of the DEVS model. The atomic model is couped to form coupled models. The structure of atomic and coupled models makes modular designing easier.

Atomic model has ports for communications and internal methods for defining the behavior. The input port receives the data sent in the bag lined to it. Whenever external input arrives, the external transition function is called to perform the required action based on the external input. Internal transition function is executed continuously based on the value of sigma. The output method is linked with the output port which pushes the output to the next interface.

Coupled model is an interface which consists of several atomic and coupled models inside. It is a set of basic components connected though the model interface. These several models interact with each other. The interaction is performed with the help of ports connected to the atomic and coupled models. The coupled models are commonly used to model complex structures that consist of multiple interconnected components. The coupling between different submodules elaborates the dependency of one module on other. Each of the consisting submodule has its own state and behavior. This is updated with the change external inputs to the module. The figure below shows the structure of coupled model.

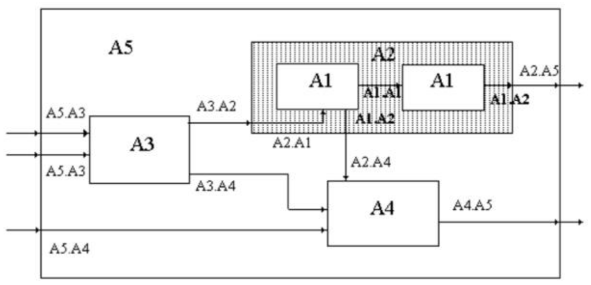


Figure 1 - diagrammatic representation of coupled model.

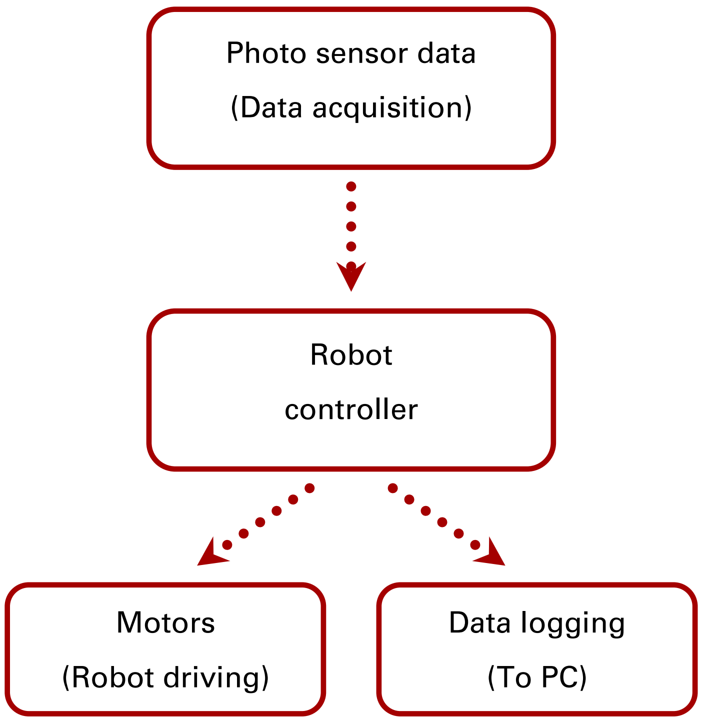
RT DEVS

The Real Time DEVS [4] is the formalism method used for modeling and simulation of real time systems that involve discrete events. It is an extension of the formalism theory of discrete event systems and provides a way to represent real time systems in terms of atomic and coupled models.

The RT-DEVS formalism is especially effective for modelling and simulating complex systems with real-time constraints, such as robotics and control systems. The formalism's flexibility makes it easy for modifications and expanding the model for future usage, and the hierarchical structure enables the system to be decomposed into smaller components that may be built and tested separately. The formalism permits simulation models to be integrated with hardware components, allowing for real-time testing and validation.

An additional benefit of using RT DEVS is that it provides a way to model and simulate systems that are difficult or impossible to test in the real world. As it is mentioned before the testing of robot behavior in hazardous environment might be risky and expensive. Simulation of these scenarios makes it easier to observe and optimize the system.

3. Development of robot model



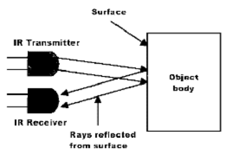
Block diagram for robot model

As a first step in the development the expected behavior of the system is defined. The robot model is described as shown above. It has four prominent contributors in the system. Their behavior is interdependent.

The block diagram is based on the problem statement of the robot development. The robot must navigate on its own. It should be able to detect the edge in the path. Based on the detected edge it should take action to rotate back and change the direction and continue the navigation. These tasks are performed on the hardware and these tasks need to be logged somewhere for understanding and analysis.

3.1 Photo sensor data acquisition

Photo sensor is a transducer. It detects the reflected light and converts it to electrical signals. It is commonly used in the application where presence or absence of light needs to be detected.



[3] Working of photo sensor.

In the case of a robot, photo sensors can be used for various purposes such as edge detection, line following, and obstacle detection. The photo sensor used for edge detection is a type of optical sensor which works on the same principle where the sensor detects light changes by measuring the intensity of the reflected light. The intensity of the reflected light changes when the sensor moves from a light surface to a dark surface, or vice versa. This change in intensity changes the value of the electrical signal which is used to detect the edge of an object. The sensor is passed through a comparator signal, and it is compared with a set threshold value to convert the analog signal to digital. This is connected to the digital pin of microcontroller.

3.2 Robot Controller

The robot motion controller block is a critical block of the edge detection robot model developed using Real-Time Discrete Event System Specification (RT-DEVS). The motion controller block is responsible for determining the behavior of the robot based on the input from the photo sensor. The block consists of several sub-components such as the sensor interface, motor controller, and control logic.

The control logic is responsible for making decisions based on the input from the sensor data and the present state of the robot. This block takes the sensor data values and determines the presence of edge. Accordingly, the action is supposed to be taken by the motor drivers which drive the wheel.

3.3 Motor driving

This Block takes the decision from the robot controller and acts accordingly. The data is sent to the drivers in the form of direction and PWM to define the direction and speed respectively. It always follows the command given by the robot controller to give specific data to specific motor.

Motor drivers are commonly used to drive motors. The motor drivers need specific commands to perform tasks. These tasks need to be defined as per requirements. The direction also needs to be defined to be decided for the motors based on the desired operation of motion. These tasks are supposed to be performed by this block.

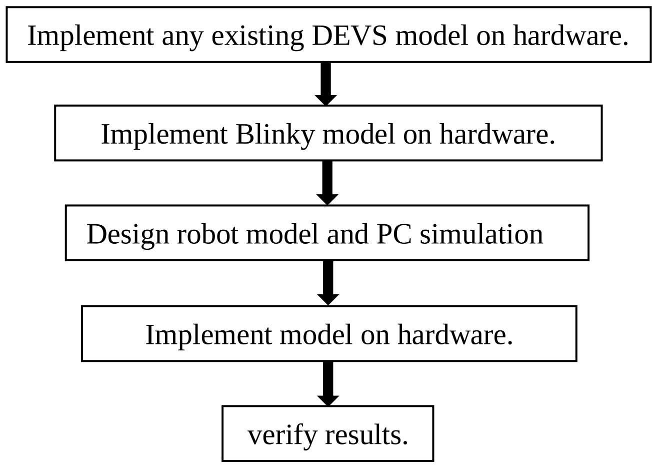
3.4 Data logging

This is an additional block considered while designing the robot application. The robot does all the simulation process on the micro controller hardware. But for observation we need the data to be displayed on the screen. The data logger dies this part. There are two applications that it must perform simultaneously. First is showing the real time data on serial terminal and second is to log the application data to some file and save the data for future use.

The data log is better done in CSV files. This is more readable and easier to sort or to work upon.

4. CONTRIBUTION

The methodology for model developemt was as follows.


Implement any existing DEVS model on hardware.
Implement Blinky model on hardware.
Design robot model and PC simulation
Implement model on hardware.
verify results.

1. Implement any existing DEVS model on hardware:

To start the project, I chose GPT model for reference and ran the model to verify the system works properly. In DEVS, a model can be designed and simulated on a computer system, but to test its real-world functionality, it is necessary to implement it on hardware. This involves implementation of hardware and flashing the model on hardware. I used STM32F411 nucleo board to test the model. Once the hardware system is ready, the model can be loaded onto it, and its performance can be observed and measured.

2. Implement Blinky model on hardware:

The Blinky model is a simple DEVS model that is commonly used for bootup purposes. It consists of two atomic models that toggle the output between two states, simulating a blinking LED. Implementing this model on hardware involves developing a system that can replicate this functionality, I used the on-board LED present on the nucleo board. The purpose of this step is to demonstrate the process of implementing a DEVS model on a hardware system in a simple and straightforward manner. Then I tried modifying the data in the model and checking the data is changing appropriately on the hardware. Here I faced some challenges for initial setup.

3. Design robot model and PC simulation:

The model shown in the previous section was developed in this step. Next to that I developed formal definitions of the atomic models and coupled models which are described in the next section. This model was then simulated on the PC for implementation verification. This step helped to make changes in the model according to the errors in simulation results.

4. Implement model on hardware:

After successful verification of model on simulation tools, the simulation was flashed on hardware. This was the step to test the functionality of edge detection in the real world. But because of certain issues it failed to work in the real world immediately. With a few changes it was fully functional. This was the crucial and time-consuming step in the development.

5. verify results:

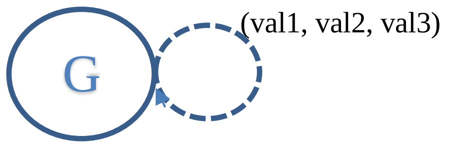
After implementing the model on hardware, it is necessary to verify the results obtained. The verification was performed on both methods, PC based simulation as well as real world simulation. The discrepancies found were analyzed and solved. The verification and documentation were the last step in the process.

5. FORMAL MODEL DEFINATIONS

The formal specifications <S, X, Y, δint, δext, λ, ta > for the atomic models are defined as follows:

Atomic: Generator

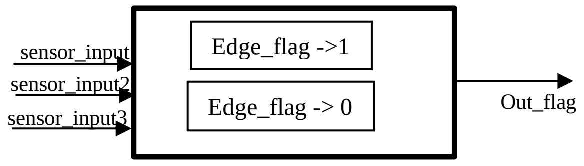
S = {G}
X = {NA}
Y = {out1, out2, out3}
δint() : δint(G) = 0.1
δext = (NA)
λ(P) = send out1 = val1
out2 = val2
out3 = val3
ta = sigma = 0.1



Atomic: motion_controller

S = {Edge_flag ==1, Edge_flag ==0}
X = { sensor_input, sensor_input2, sensor_input3}
Y = {Out_flag}

δint() : sigma = infinite
δext() :
if(sensor_input == 1)
edge_flag = 1;
else if(sensor_input2 == 1)
edge_flag = 1;
else if(sensor_input3 == 1)
edge_flag = 1;
λ() = send
out_flag = edge_flag
ta = sigma



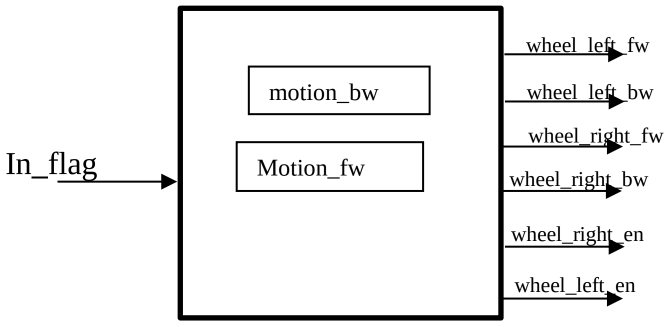
Motion controller atomic model

Atomic: wheel

S = {motion_bw, Motion_fw}
X = { In_Flag }
Y = { wheel_left_fw, wheel_left_bw, wheel_right_fw, wheel_right_bw, wheel_right_en, wheel_left_en }

δint() : sigma = infinite
δext() : edge_flag = in_flag;
if(s.Edge_flag == 0)
sigma = DrivePeriod*3;
else
sigma = DrivePeriod;

λ() = right_wheel = enable;
left_wheel = enable;
if((s.Edge_flag == 0))
backward_motion();
else
forward_motion();



Coupled: Edge_robot

X = {};

Y = {}

D = {wheel, motion_controller, generator, digitalSenseInput1, digitalSenseInput2, digitalSenseInput3, PWMOutput1, PWMOutput2, PWMOutput3, PWMOutput4, MotorEnableLeft, MotorEnableRight};

EIC = {};

EOC = {};

If (RT_ARM_MBED)

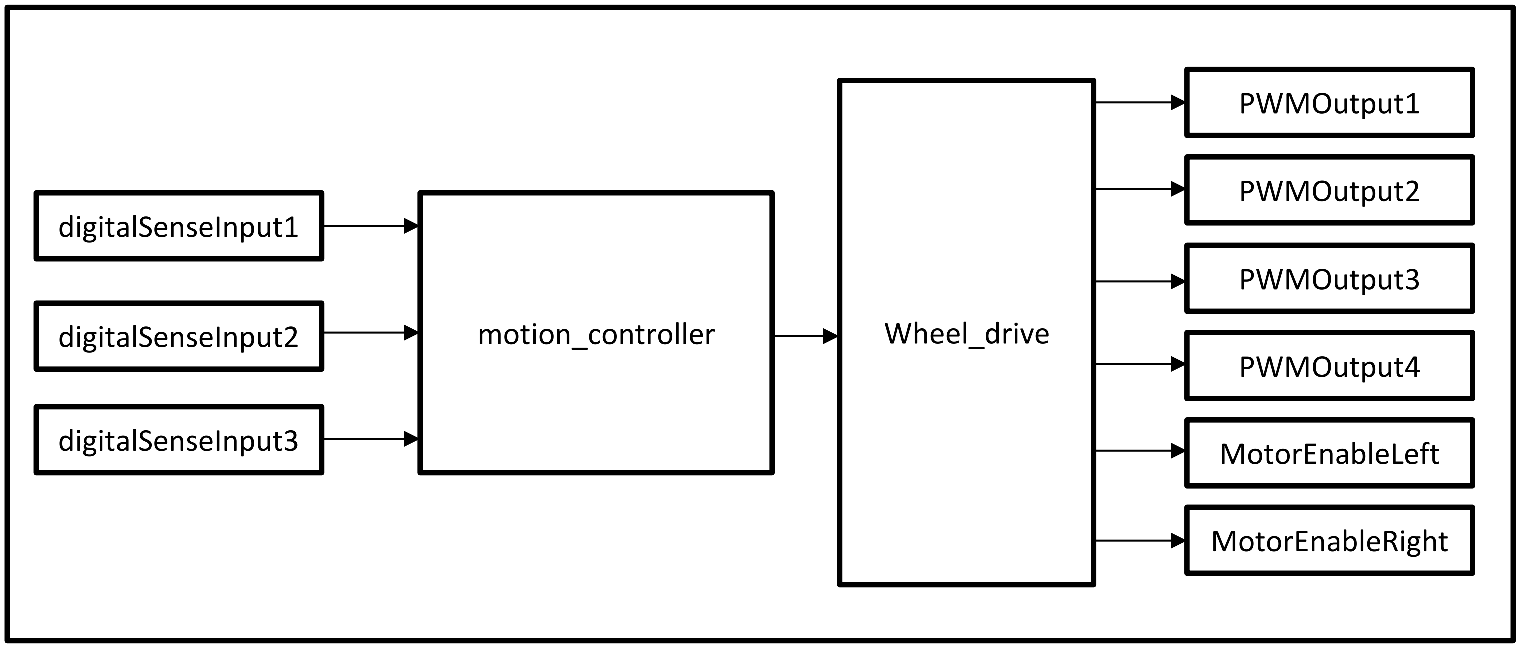
IC = { (digitalSenseInput1->out, motion_controller->sensor_input), (digitalSenseInput2->out, motion_controller->sensor_input2), (digitalSenseInput3->out, motion_controller->sensor_input3), (wheel_drive->wheel_left_fw, PWMOutput1->in), (wheel_drive->wheel_left_bw, PWMOutput2->in), (wheel_drive->wheel_right_fw, PWMOutput3->in), (wheel_drive->wheel_right_bw, PWMOutput4->in), (wheel_drive->wheel_left_en, MotorEnableLeft->in), (wheel_drive->wheel_right_en, MotorEnableRight->in)

Else

IC = {(generator->out1, motion_controller->sensor_input), (generator->out2, motion_controller->sensor_input2), (generator->out3, motion_controller->sensor_input3),}

Common

IC = {(motion_controller->Out_flag, wheel_drive->In_Flag)}

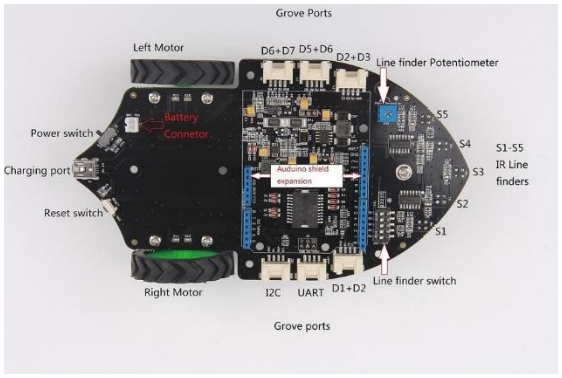


Edge Robot Coupled Model

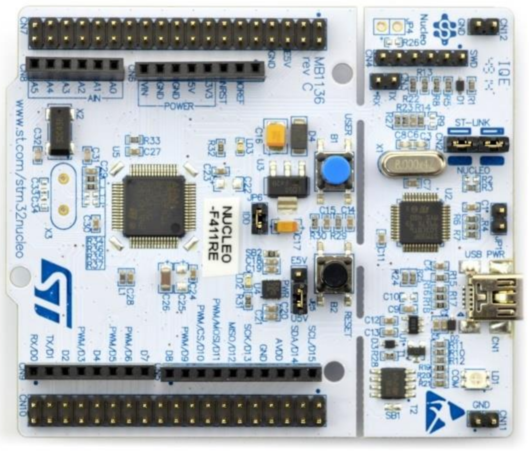
6. RESULTS

Hardware setup –

1. Robot shield –



2. Microcontroller used - STM32F411RE

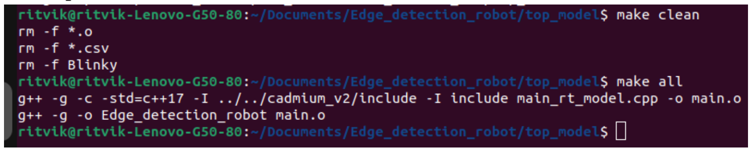


Other tools –

3. Cadmium V2 (https://github.com/epecker/cadmium_v2)
4. Lipo Batter – 7V
5. PuTTY (Release 0.78)
6. Mbed OS

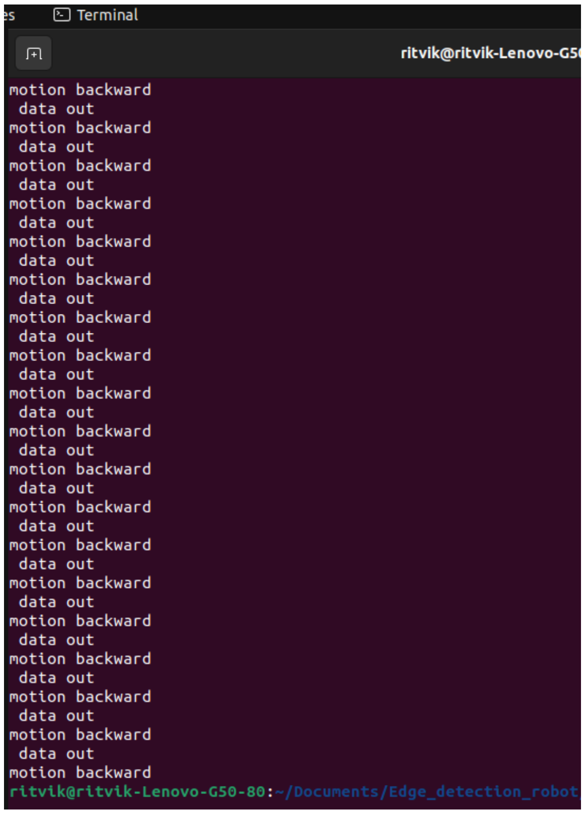
PC Simulations –

Compilation in PC



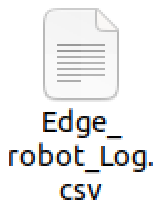
```
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$ make clean
rm -f *.o
rm -f *.csv
rm -f Blinky
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$ make all
g++ -g -c -std=c++17 -I ../../cadmium_v2/include -I include main_rt_model.cpp -o main.o
g++ -g -o Edge_detection_robot main.o
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$
```

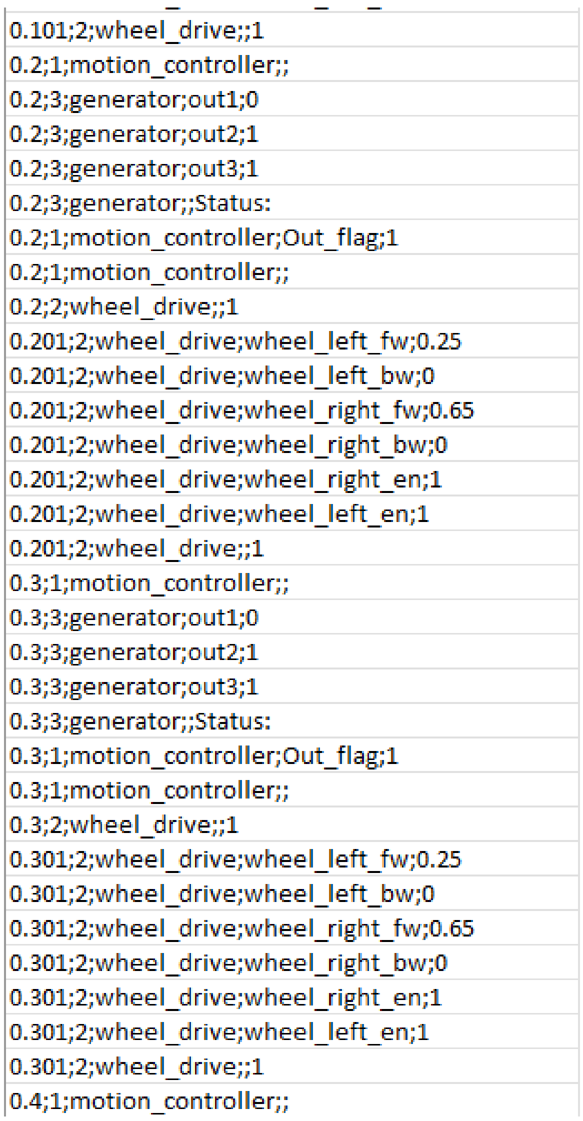
Running the output file



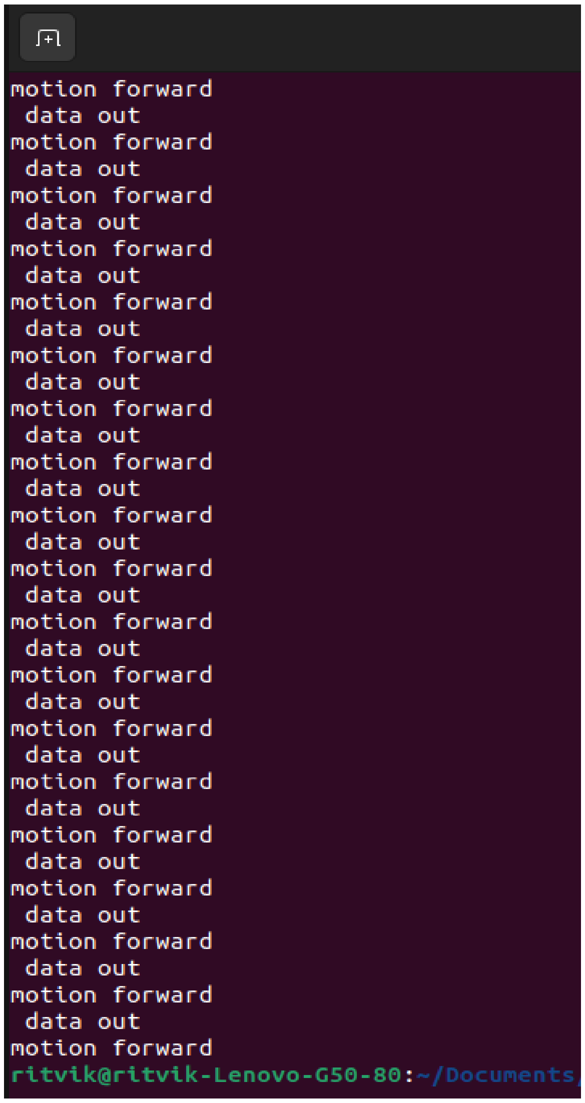
```
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
 data out
motion backward
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot
```

The simulation results obtained in csv





```
0.101;2;wheel_drive;;1
0.2;1;motion_controller;;
0.2;3;generator;out1;0
0.2;3;generator;out2;1
0.2;3;generator;out3;1
0.2;3;generator;;Status:
0.2;1;motion_controller;Out_flag;1
0.2;1;motion_controller;;
0.2;2;wheel_drive;;1
0.201;2;wheel_drive;wheel_left_fw;0.25
0.201;2;wheel_drive;wheel_left_bw;0
0.201;2;wheel_drive;wheel_right_fw;0.65
0.201;2;wheel_drive;wheel_right_bw;0
0.201;2;wheel_drive;wheel_right_en;1
0.201;2;wheel_drive;wheel_left_en;1
0.201;2;wheel_drive;;1
0.3;1;motion_controller;;
0.3;3;generator;out1;0
0.3;3;generator;out2;1
0.3;3;generator;out3;1
0.3;3;generator;;Status:
0.3;1;motion_controller;Out_flag;1
0.3;1;motion_controller;;
0.3;2;wheel_drive;;1
0.301;2;wheel_drive;wheel_left_fw;0.25
0.301;2;wheel_drive;wheel_left_bw;0
0.301;2;wheel_drive;wheel_right_fw;0.65
0.301;2;wheel_drive;wheel_right_bw;0
0.301;2;wheel_drive;wheel_right_en;1
0.301;2;wheel_drive;wheel_left_en;1
0.301;2;wheel_drive;;1
0.4;1;motion_controller;;
```



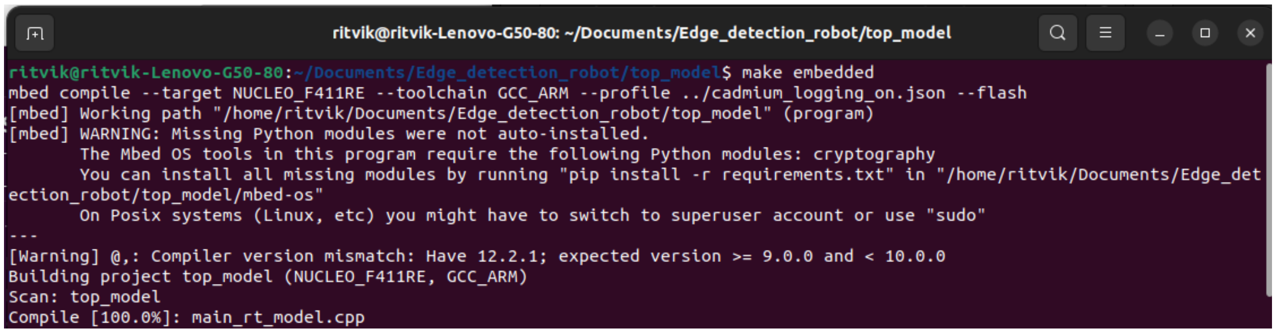
```
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
 data out
motion forward
ritvik@ritvik-Lenovo-G50-80:~/Documents
```

Observation table

Sensor value 1	Sensor value 2	Sensor value 3	Left_wheel_Fw	Left_wheel_Bw	Right_wheel_Fw	Right_wheel_Bw
0	0	0	0.25	0	0.25	0
0	0	1	0	0.25	0	0.65
0	1	0	0	0.25	0	0.65
0	1	1	0	0.25	0	0.65
1	0	0	0	0.25	0	0.65
1	0	1	0	0.25	0	0.65
1	1	0	0	0.25	0	0.65
1	1	1	0	0.25	0	0.65

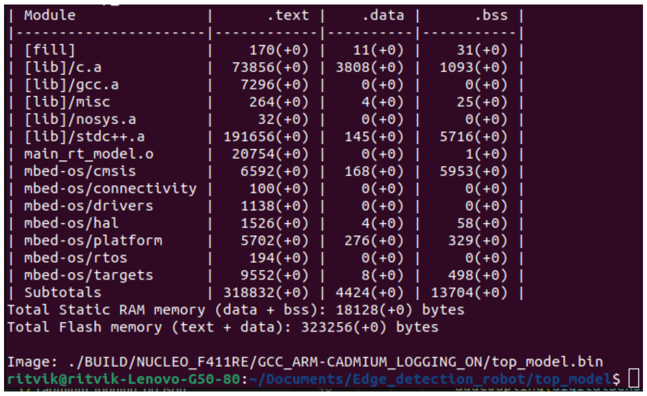
Real Time Simulation –

Compilation



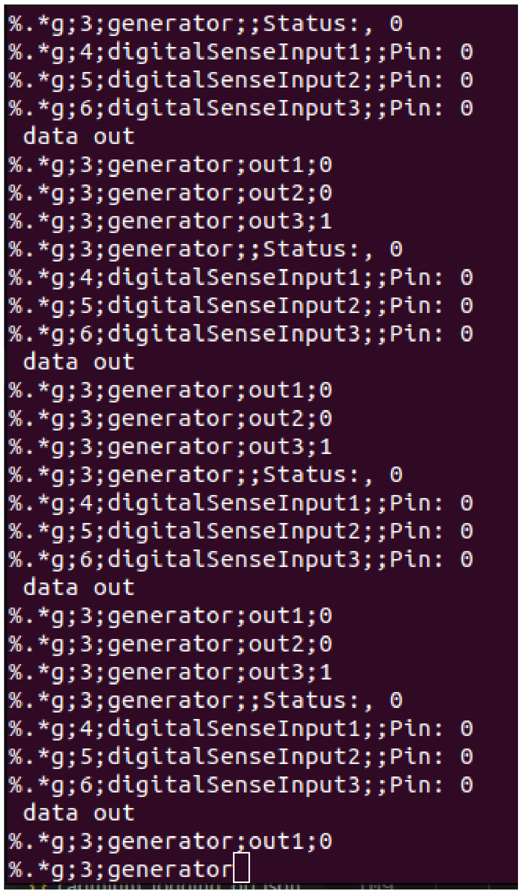
```
ritvik@ritvik-Lenovo-G50-80: ~/Documents/Edge_detection_robot/top_model
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$ make embedded
mbed compile --target NUCLEO_F411RE --toolchain GCC_ARM --profile ../cadmium_logging_on.json --flash
[mbed] Working path "/home/ritvik/Documents/Edge_detection_robot/top_model" (program)
[mbed] WARNING: Missing Python modules were not auto-installed.
       The Mbed OS tools in this program require the following Python modules: cryptography
       You can install all missing modules by running "pip install -r requirements.txt" in "/home/ritvik/Documents/Edge_det
ection_robot/top_model/mbed-os"
       On Posix systems (Linux, etc) you might have to switch to superuser account or use "sudo"
---
[Warning] @,: Compiler version mismatch: Have 12.2.1; expected version >= 9.0.0 and < 10.0.0
Building project top_model (NUCLEO_F411RE, GCC_ARM)
Scan: top_model
Compile [100.0%]: main_rt_model.cpp
```

Flashing



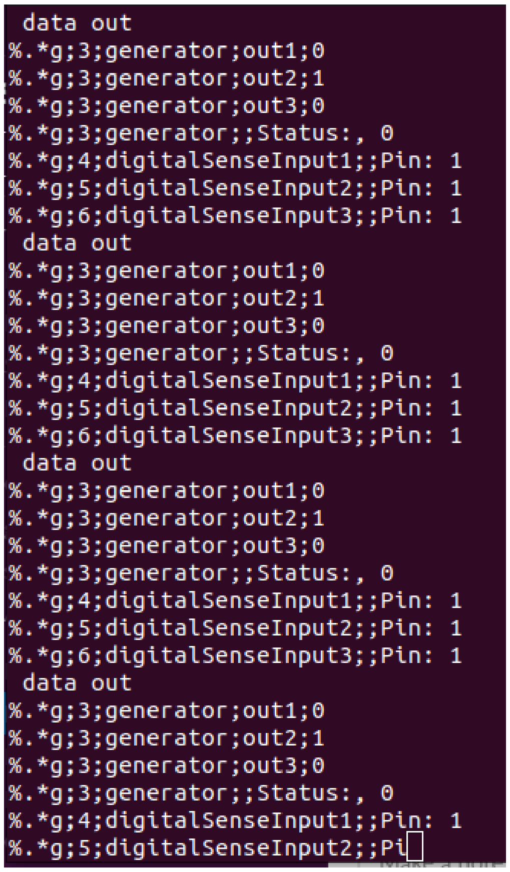
```
| Module                | .text      | .data     | .bss       |
|-----------------------|------------|-----------|------------|
| [fill]                |    170(+0) |    11(+0) |     31(+0) |
| [lib]/c.a             |  73856(+0) |  3808(+0) |   1093(+0) |
| [lib]/gcc.a           |   7296(+0) |     0(+0) |      0(+0) |
| [lib]/misc            |    264(+0) |     4(+0) |     25(+0) |
| [lib]/nosys.a         |     32(+0) |     0(+0) |      0(+0) |
| [lib]/stdc++.a        | 191656(+0) |   145(+0) |   5716(+0) |
| main_rt_model.o       |  20754(+0) |     0(+0) |      1(+0) |
| mbed-os/cmsis         |   6592(+0) |   168(+0) |   5953(+0) |
| mbed-os/connectivity  |    100(+0) |     0(+0) |      0(+0) |
| mbed-os/drivers       |   1138(+0) |     0(+0) |      0(+0) |
| mbed-os/hal           |   1526(+0) |     4(+0) |     58(+0) |
| mbed-os/platform      |   5702(+0) |   276(+0) |    329(+0) |
| mbed-os/rtos          |    194(+0) |     0(+0) |      0(+0) |
| mbed-os/targets       |   9552(+0) |     8(+0) |    498(+0) |
| Subtotals             | 318832(+0) |  4424(+0) |  13704(+0) |
Total Static RAM memory (data + bss): 18128(+0) bytes
Total Flash memory (text + data): 323256(+0) bytes

Image: ./BUILD/NUCLEO_F411RE/GCC_ARM-CADMIUM_LOGGING_ON/top_model.bin
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$
```

All sensors outside



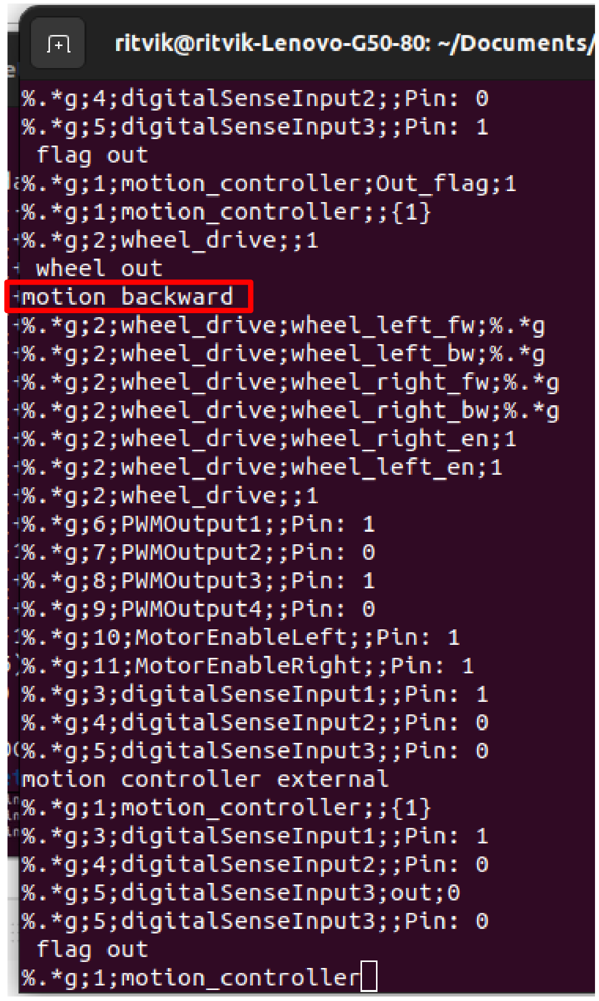
```
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 0
%.*g;5;digitalSenseInput2;;Pin: 0
%.*g;6;digitalSenseInput3;;Pin: 0
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;0
%.*g;3;generator;out3;1
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 0
%.*g;5;digitalSenseInput2;;Pin: 0
%.*g;6;digitalSenseInput3;;Pin: 0
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;0
%.*g;3;generator;out3;1
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 0
%.*g;5;digitalSenseInput2;;Pin: 0
%.*g;6;digitalSenseInput3;;Pin: 0
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;0
%.*g;3;generator;out3;1
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 0
%.*g;5;digitalSenseInput2;;Pin: 0
%.*g;6;digitalSenseInput3;;Pin: 0
 data out
%.*g;3;generator;out1;0
%.*g;3;generator
```

All sensors inside



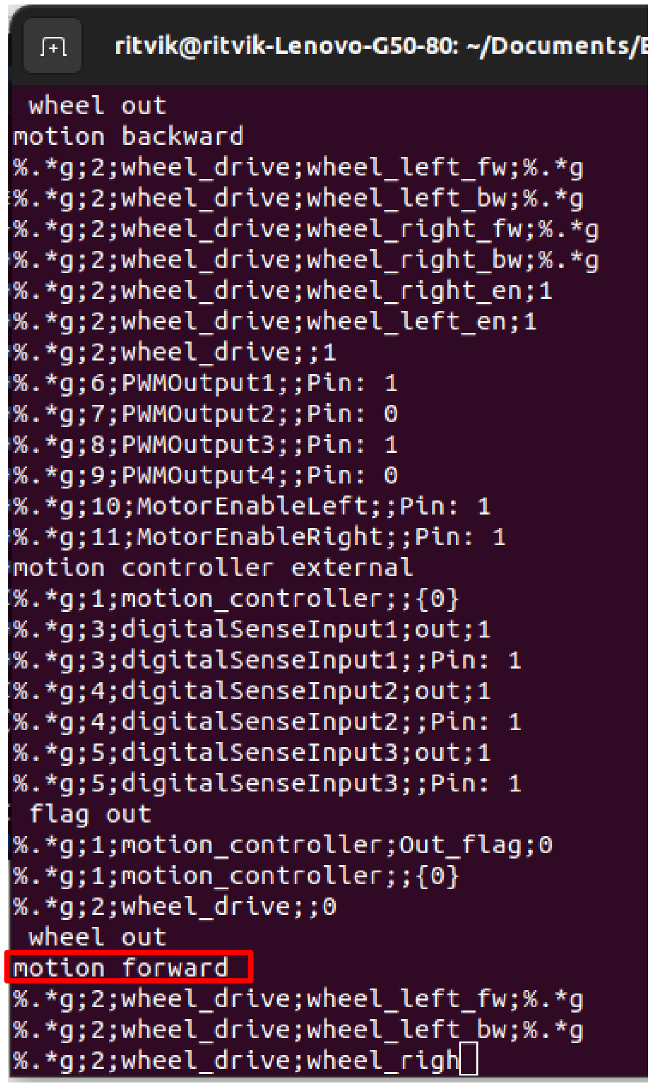
```
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;1
%.*g;3;generator;out3;0
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 1
%.*g;5;digitalSenseInput2;;Pin: 1
%.*g;6;digitalSenseInput3;;Pin: 1
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;1
%.*g;3;generator;out3;0
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 1
%.*g;5;digitalSenseInput2;;Pin: 1
%.*g;6;digitalSenseInput3;;Pin: 1
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;1
%.*g;3;generator;out3;0
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 1
%.*g;5;digitalSenseInput2;;Pin: 1
%.*g;6;digitalSenseInput3;;Pin: 1
 data out
%.*g;3;generator;out1;0
%.*g;3;generator;out2;1
%.*g;3;generator;out3;0
%.*g;3;generator;;Status:, 0
%.*g;4;digitalSenseInput1;;Pin: 1
%.*g;5;digitalSenseInput2;;Pi
```

Transition (Edge detection)



```
ritvik@ritvik-Lenovo-G50-80: ~/Documents/
%.*g;4;digitalSenseInput2;;Pin: 0
%.*g;5;digitalSenseInput3;;Pin: 1
 flag out
%.*g;1;motion_controller;Out_flag;1
%.*g;1;motion_controller;;{1}
%.*g;2;wheel_drive;;1
 wheel out
motion backward
%.*g;2;wheel_drive;wheel_left_fw;%.*g
%.*g;2;wheel_drive;wheel_left_bw;%.*g
%.*g;2;wheel_drive;wheel_right_fw;%.*g
%.*g;2;wheel_drive;wheel_right_bw;%.*g
%.*g;2;wheel_drive;wheel_right_en;1
%.*g;2;wheel_drive;wheel_left_en;1
%.*g;2;wheel_drive;;1
%.*g;6;PWMOutput1;;Pin: 1
%.*g;7;PWMOutput2;;Pin: 0
%.*g;8;PWMOutput3;;Pin: 1
%.*g;9;PWMOutput4;;Pin: 0
%.*g;10;MotorEnableLeft;;Pin: 1
%.*g;11;MotorEnableRight;;Pin: 1
%.*g;3;digitalSenseInput1;;Pin: 1
%.*g;4;digitalSenseInput2;;Pin: 0
%.*g;5;digitalSenseInput3;;Pin: 0
motion controller external
%.*g;1;motion_controller;;{1}
%.*g;3;digitalSenseInput1;;Pin: 1
%.*g;4;digitalSenseInput2;;Pin: 0
%.*g;5;digitalSenseInput3;out;0
%.*g;5;digitalSenseInput3;;Pin: 0
 flag out
%.*g;1;motion_controller
```

Transition 2 (Back to plane)



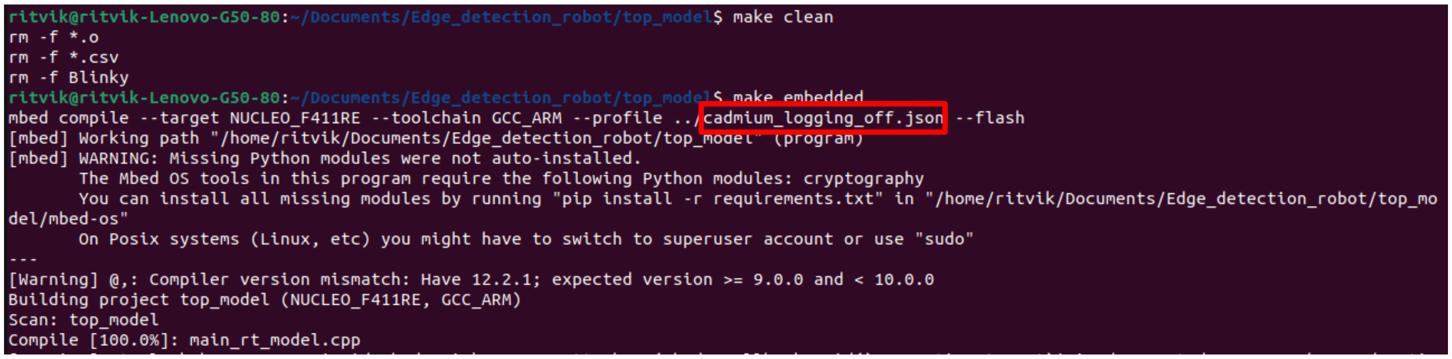
```
ritvik@ritvik-Lenovo-G50-80: ~/Documents/
 wheel out
motion backward
%.*g;2;wheel_drive;wheel_left_fw;%.*g
%.*g;2;wheel_drive;wheel_left_bw;%.*g
%.*g;2;wheel_drive;wheel_right_fw;%.*g
%.*g;2;wheel_drive;wheel_right_bw;%.*g
%.*g;2;wheel_drive;wheel_right_en;1
%.*g;2;wheel_drive;wheel_left_en;1
%.*g;2;wheel_drive;;1
%.*g;6;PWMOutput1;;Pin: 1
%.*g;7;PWMOutput2;;Pin: 0
%.*g;8;PWMOutput3;;Pin: 1
%.*g;9;PWMOutput4;;Pin: 0
%.*g;10;MotorEnableLeft;;Pin: 1
%.*g;11;MotorEnableRight;;Pin: 1
motion controller external
%.*g;1;motion_controller;;{0}
%.*g;3;digitalSenseInput1;out;1
%.*g;3;digitalSenseInput1;;Pin: 1
%.*g;4;digitalSenseInput2;out;1
%.*g;4;digitalSenseInput2;;Pin: 1
%.*g;5;digitalSenseInput3;out;1
%.*g;5;digitalSenseInput3;;Pin: 1
 flag out
%.*g;1;motion_controller;Out_flag;0
%.*g;1;motion_controller;;{0}
%.*g;2;wheel_drive;;0
 wheel out
motion forward
%.*g;2;wheel_drive;wheel_left_fw;%.*g
%.*g;2;wheel_drive;wheel_left_bw;%.*g
%.*g;2;wheel_drive;wheel_righ
```

The tests were executed using two setup –

1. With Cadmium logging on
2. With Cadmium logging off

The Cadmium logging on was done to capture the trace of real time log of the simulation. But as per the observation, the logging function APIs are blocking. When you call those APIs, they add extra delay in the execution. The real time applications are time critical. This delay causes errors in the executions. It does not allow the real time task to follow the time bound. To tackle this, I followed another method in which the logging function was turned off and the execution was performed on the hardware. Initially when the robot was tested with the edge detection functionality, the robot failed to achieve the same with the logging. Unit case testing was successful. The results were theoretically correct but delayed in time. When the second binary with logging off was uploaded the edge detection in real world environment was successful.

Compilation with logging off.



```
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$ make clean
rm -f *.o
rm -f *.csv
rm -f Blinky
ritvik@ritvik-Lenovo-G50-80:~/Documents/Edge_detection_robot/top_model$ make embedded
mbed compile --target NUCLEO_F411RE --toolchain GCC_ARM --profile ../cadmium_logging_off.json --flash
[mbed] Working path "/home/ritvik/Documents/Edge_detection_robot/top_model" (program)
[mbed] WARNING: Missing Python modules were not auto-installed.
       The Mbed OS tools in this program require the following Python modules: cryptography
       You can install all missing modules by running "pip install -r requirements.txt" in "/home/ritvik/Documents/Edge_detection_robot/top_mo
del/mbed-os"
       On Posix systems (Linux, etc) you might have to switch to superuser account or use "sudo"
---
[Warning] @,: Compiler version mismatch: Have 12.2.1; expected version >= 9.0.0 and < 10.0.0
Building project top_model (NUCLEO_F411RE, GCC_ARM)
Scan: top_model
Compile [100.0%]: main_rt_model.cpp
```

7. CONCLUSION

In conclusion, this paper presented a Real-Time DEVS-based model for an edge detection robot that was implemented using the Real Time Cadmium simulation environment. The model consists of multiple atomic models such as motor controllers, wheel drive, and logic for motion control. The simulation results showed that the robot was able to successfully detect edges and move accordingly. The results also show the effectiveness of the developed model in various scenarios in which the robot was able to give correct output.

The model shown in the paper can be further used for the implementation of advanced control algorithms for the robot with extension to the current implementation like incorporating other sensors for object detection and extending the model for multi-robot systems. Also, we concluded that the real-time capabilities of the DEVS formalism can be used for the development of more complex and critical systems in various domains, such as autonomous vehicles, robotics, and aerospace.

The simulation model of the edge detection robot using RT-DEVS presented in this paper has demonstrated its ability to simulate the behavior of a complex real-time system effectively and unambiguously. The modular and hierarchical structure of DEVS supported us for the development of a flexible and scalable model. An additional advantage is that it can be easily modified and extended for further research and experimentation. The simulation results have also shown that the proposed edge detection algorithm is effective for the task of detecting edge and avoiding it by taking actions.

This paper also demonstrates that the simulation-based approach presented in this paper can be a cost-effective and safe alternative to physical testing and experimentation, especially for risky and hazardous tasks.

Overall, this paper has contributed to the field of robotics by demonstrating the application of DEVS formalism for the development of real-time robot simulation models. This paper has highlighted the potential of RT-DEVS as a powerful tool for the modeling and simulation of various real-time systems, including robotics, and other control systems. The research presented in this paper can be extended to explore other aspects of robot behavior, such as motion planning, pathfinding, and obstacle avoidance.

8. FUTURE WORK

- The presented work in this paper is a contribution to the field of edge detection robotics using Real-Time DEVS formalism. However, there are still many areas that can be explored to enhance the model's functionality and performance by performing future research and development.

- The model proposed in the research can be extended to implement additional features such as avoiding obstacles, which can help the robot navigate through complex environments safely. The way to achieve this can be by incorporating additional sensors to the model and integrating the functionality logic to the simulation.

- Model can be further optimized for real-time applications after improving the performance of algorithms and hardware. This will enable the robot to operate smoothly and with more accuracy. This will also help in improvising the overall throughput.

- The given model can also be extended to support swarm robots. For this implementation we can enable the simulation to make the robots work collaboratively and perform tasks that are beyond the capabilities of a single robot. This will require development of communication protocols inside the simulation for exchange of data.

- The proposed model can be used as a basis for developing better featured edge detection robots that can operate in various other environments and perform more secondary tasks. We can also incorporate machine learning algorithms to enable the robot to learn and adapt its behavior according to the environment.

- Edge detection is an important task in computer vision, and there is a lot of exposure to use image processing algorithms to improve accuracy and efficiency. You could explore how the DEVS formalism methods could be integrated with machine learning algorithms to improve the robot's ability to detect edges or perform various robotic tasks.

- The DEVS model developed during the project was implemented on a small-scale robot. There is potential to scale it up for use in larger robots or even industrial automation systems. The challenges and opportunities involved in scaling up the model and investigating can be explored, and it could be used in a variety of real-world applications.
- Overall, the proposed model has significant potential research and development in the future and can be improved and optimized to meet the evolving needs of the field of edge detection robotics.

9. REFERENCES

[1] Concepcion, A.I., and B.P. Zeigler. 1988. “DEVS Formalism: A Framework for Hierarchical Model

[2] Earle, B., K. Bjornson, C. Ruiz-Martin, and G. Wainer. 2020. “Development of a Real-Time Devs Kernel: Rt-Cadmium.”. In Spring Simulation Conference (SpringSim 2020), pp. 1–12. Fairfax, VA.

[3] G L.Boaz1, S.Priyatharshini2, G.Bharatha Sreeja3 , R.Lavanya: AVR Microcontroller Based Obstacle Monitoring Console for Automobiles and Industrial Automation. *International Journal of Innovative Research in Computer and Communication Engineering Vol. 4, Special Issue 2, April 2016*

[4] Hong, J. S., H. S. Song, T. G. Kim, and K. H. Park. 1997. “A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development.”. Discrete Event Dynamic Systems vol. 7, no. 4, pp. 355–75. https://doi.org/10.1023/a:1008262409521.

[5] Belloli, L., D. Vicino, C. Ruiz-Martin, and G. Wainer. 2019. “Building Devs Models with the Cadmium Tool”. In Proceedings of the 2019 Winter Simulation Conference, pp. 45–59. National Harbor, Maryland.

[6] Furfaro, A., Nigro, L. A development methodology for embedded systems based on RT-DEVS. Innovations Syst Softw Eng 5, 117–127 (2009). https://doi.org/10.1007/s11334-009-0085-4

[7] Rami Al-Jarrah, Mohammad Al-Jarrah, and Hubert Roth. A Novel Edge Detection Algorithm for Mobile Robot Path Planning. Journal of Robotics. Volume 2018, https://doi.org/10.1155/2018/1969834

[8] Angelo Furfaro · Libero Nigro. A development methodology for embedded systems based on RT-DEVS. Innovations Syst Softw Eng (2009) 5:117–127

[9] Wainer, G. 2009. Discrete-Event Modeling and Simulation. Boca Raton, FL, USA: CRC Press.

[10] Zeigler, B. P. 1989. “DEVS Representation of Dynamical Systems: Event-Based Intelligent Control.”. Proceedings of the IEEE vol. 77, no. 1, pp. 72–80. https://doi.org/10.1109/5.21071.

[11] Wainer, G. 2009. Discrete-Event Modeling and Simulation: A Practitioner’s Approach (Computational Analysis, Synthesis, and Design of Dynamic Systems). CRC Press, Inc.

[12] Zeigler, B., H. Prähofer, and T.G. Kim. 2000. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press. San Diego, CA.