University of California, Berkeley

ME 235: Design of Microprocessors

Autonomous Lidar Mapping

Team members :
C.K Wolfe
Holiday Randriama
Vianney Grenez
Farouk Mostafa

Teachers :
George Anwar

Contents

TODO: grammar edit for all sections

Acknowledgements

The development process was an altogether enjoyable experience thanks in part to the interactions with the professors and GSI's through the debugging and iterative design process. Thanks for your patience, assistance, and sense of humor. We are proud to present the final results.

1 Background & Motivation

The motivation for creating this project is to help develop and test the code for an Electric Vehicle Kart using a $\frac{1}{10}$ scale RC car as a model. We wish to be able to implement the autonomous feature onto the full scale EV Kart which will be run during the Electric Vehicle Grand Prix in West Lafayette, Indiana. A collective collaboration under the AI Racing Tech umbrella with the University of Hawaii, the University of California, San Diego, and the University of California, Berkeley, set out to create the full scale EV Kart and currently uses autonomous $\frac{1}{10}$ scale RC as a reference to help develop it. Small scale testing allows the team to troubleshoot and debug pipeline issues when it comes to implementing perception solutions, and explore the functionality and accuracy beyond the simulation.

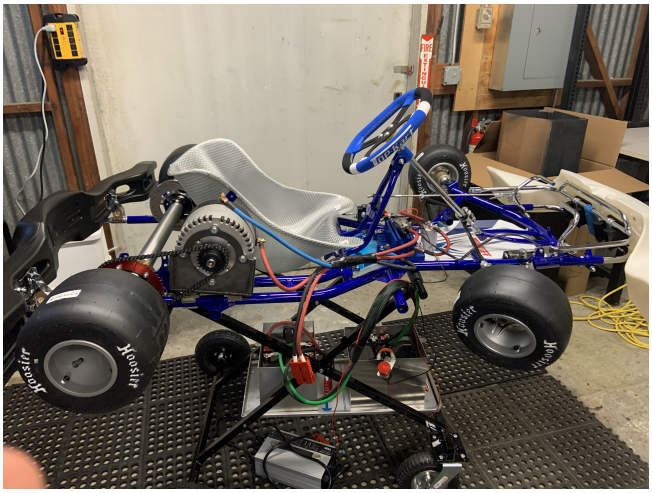


Figure 1: Autonomous eVKart in progress

Figure 2: **Modular testing with small scale RC chassis base** used to configure experimental sensor hardware and experimental software, used test controls and perception software in real-world conditions

2 The Overall Structure of the Car

The final design of this project was a path following autonomous vehicle that explored the feasibility of IR sensors and LiDAR based SLAM (Simultaneous Localization and Mapping) for perception based localization. The Robot successfully navigated a path using perception based localization only, it was able to map it's surroundings using 2D 360 degree LiDAR. The project primarily used a Raspberry Pi for BLE communication and LDS-01 LiDAR management, a PSOC6 for real-time controls and IR sensor management as well as integration with a LabVIEW GUI. The Robot was able to simultaneously transmit and receive massages from two different PCs. The data flow between components is shown in the Flowchart below in Figure 5. This report will walk you through the development steps and the challenges faced in implementing this final project, as well as discuss areas of future improvement.

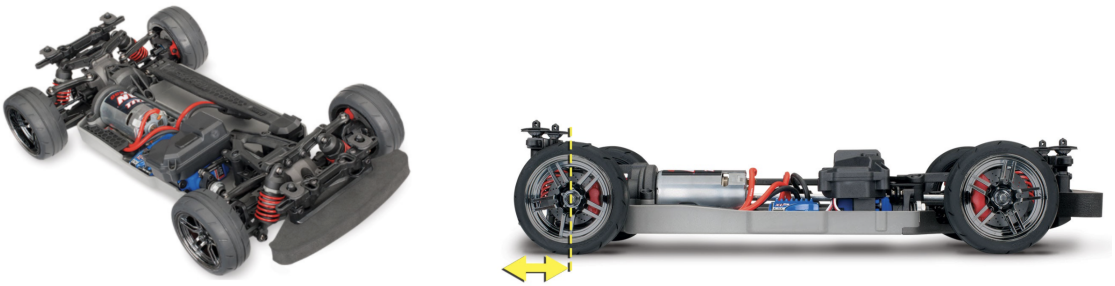


Figure 3: Traxxas Chassis, the Experimental "Blank Slate"
Side-view of Traxxas $\frac{1}{10}$ Scale 4-Tec 2.0 AWD Chassis, capable of achieving up to 50mph

The LiDAR sensor purchased for this project is shown below, it is a single line scan that allows 2D mapping only. The resolution is better at slower speeds. Higher resolution LiDAR sensors were prohibitively expensive this project, this lower cost implementation still allowed for software implementation and an overall proof of concept.

Figure 4: Purchased for this project, LDS-01 360 2D LiDAR sensor, Robotis

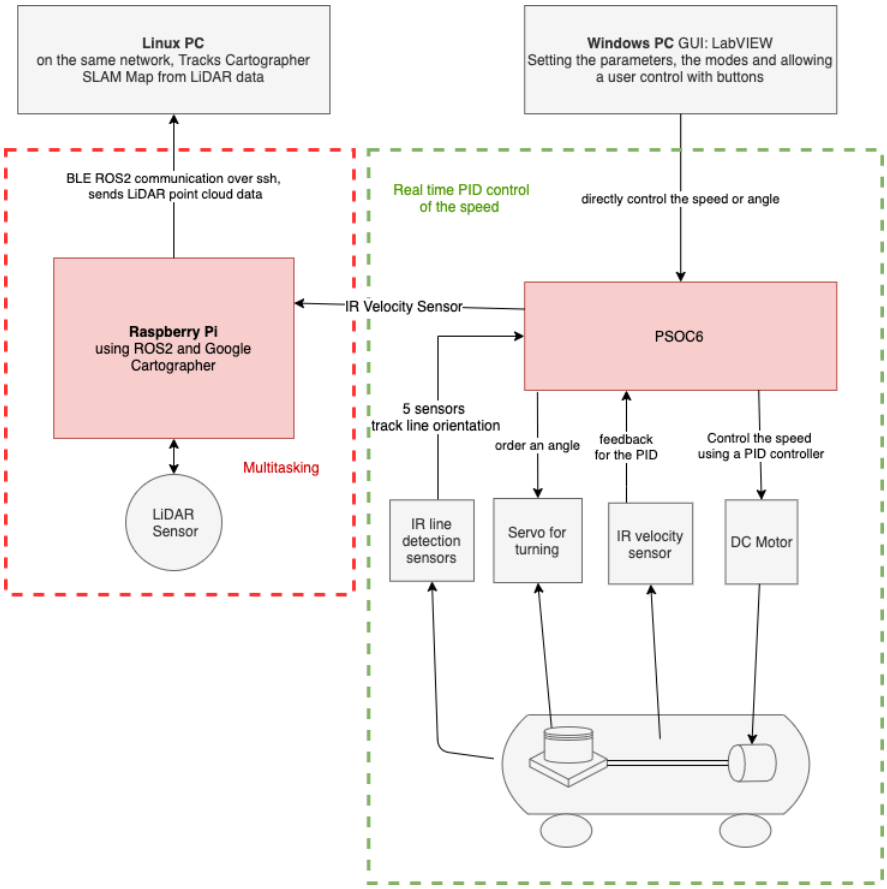


Figure 5: Flowchart for Hardware and Software Integration

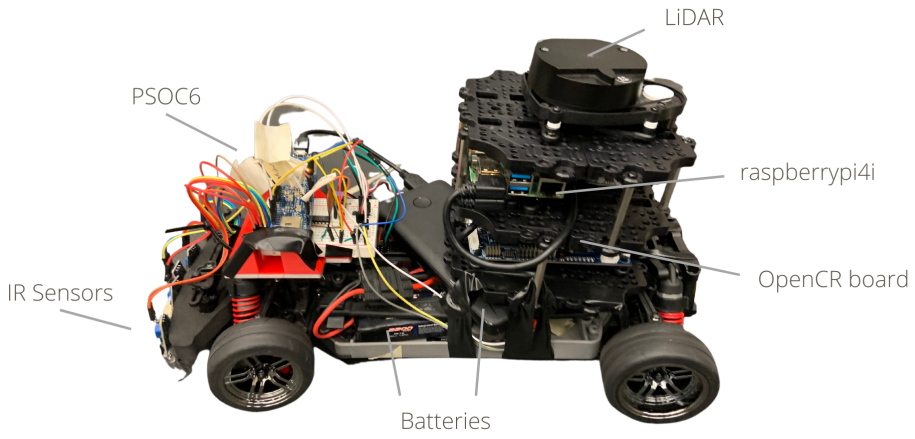


Figure 6: Final Car Assembly, displaying the experimental sensor package this team designed

3 The Graphical User Interface

3.1 Presentation

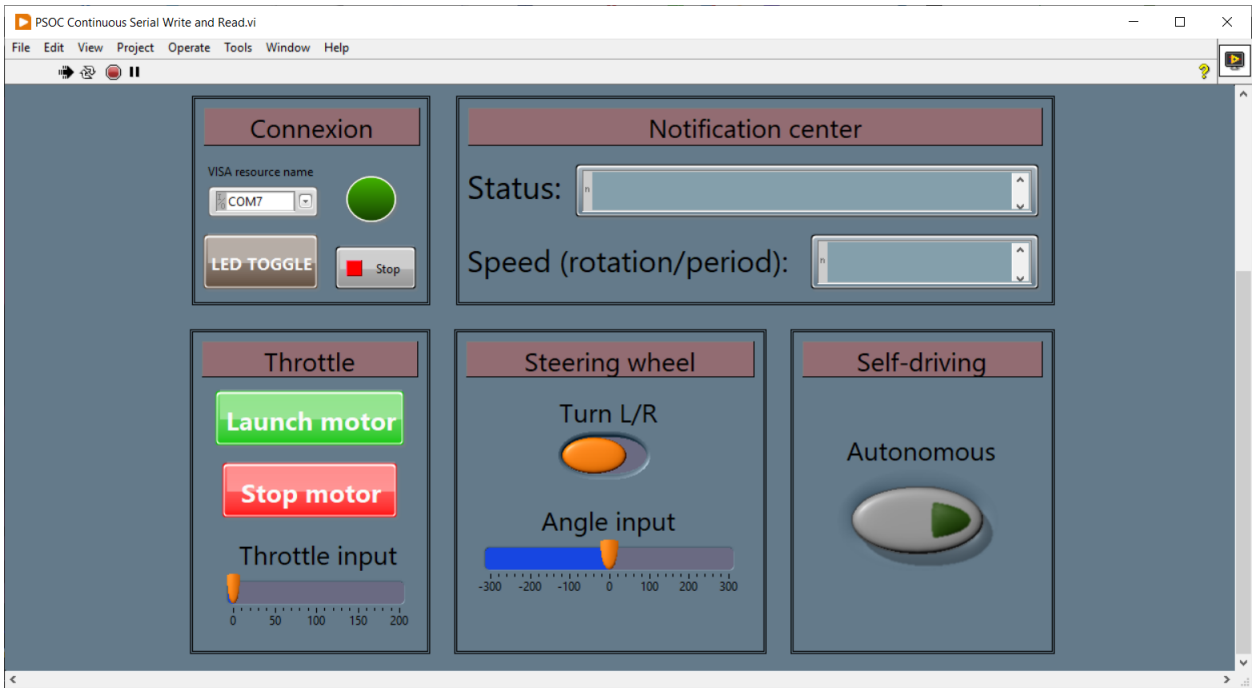


Figure 7: LabVIEW graphical user interface

The user interface was meticulously designed on LabVIEW to make the robot as easy to control as possible. This usability was reached by introducing five different sections:

- **Connection**: establish serial communication between LabVIEW and the PSoC
- **Notifications center**: receive the status of the robot, or the actions that the robot is currently taking
- **Throttle**: Launch or stop the motor, or set the motor to a specific speed
- **Steering Wheel**: Steer the wheel left, right, or in a specific angle
- **Self driving**: set the car to autonomously steer the car to follow a line

3.2 Sending instructions to the Robot via PSOC6

As the master in the master/slave UART protocol, the GUI is the one that sends all instructions to the robot, either for performing a specific action or to receive the speed of the car.

On the user interface, each button is associated with a unique string to be sent. As an example, the **'Launch motor'** button is associated to the character 'A', and the horizontal pointer slide corresponding to the **'Steering angle'** is associated to the string 'V'

+ the value of the steering angle.

Two different types of events are handled to send instructions:

- Button clicks, or sliders value change, to actuate specific instructions to the car
- A specific timed loop, to asks for the value of the RC car speed each 1000 ms

This implementation allows the GUI to handle both the user's interaction with the buttons and sliders, and an automatic monitoring of the car's speed

3.3 Communication protocol

Communication between the UART and the PSoC is established by an UART protocol, where the GUI acts as a master and the robot as a slave. That means the serial communication is used only for the GUI to send instructions and for the robot to sent its status after receiving instructions.

The figure below explains this protocol and displays the 6 different communication states on the GUI side:

- **Init**: Initialize the UART characteristics
- **Connect**: Establish the connection between the GUI and the PSoC
- **Idle**: The GUI is in an idle mode when connection has been established and there is nothing to send
- **Send**: When an instruction is to be sent, the GUI is in write mode and sends the appropriate instruction to the PSoC
- **Wait**: After sending an instruction, the GUI waits for the PSoC's answer
- **Receive**: When bytes are received on from the PSoC, the GUI processes the received message and prints the status of the robot

As the RC car was expected to follow a line, we first imagined a wireless communication between the GUI and our PSoC micro-controller. We chose the PSoC 063 chip especially because of the bluetooth BLE feature that would provide this wireless connection. Unfortunately, the implementation of the BLE via PSOC was revealed to be harder than expected, as we will discuss at the end of this section. BLE communication was ultimately executed via the Raspberry PI for the LiDAR sensor, and we ultimately preferred this BLE implementation approach over the PSoC as it offered increased speed and easier implementation.

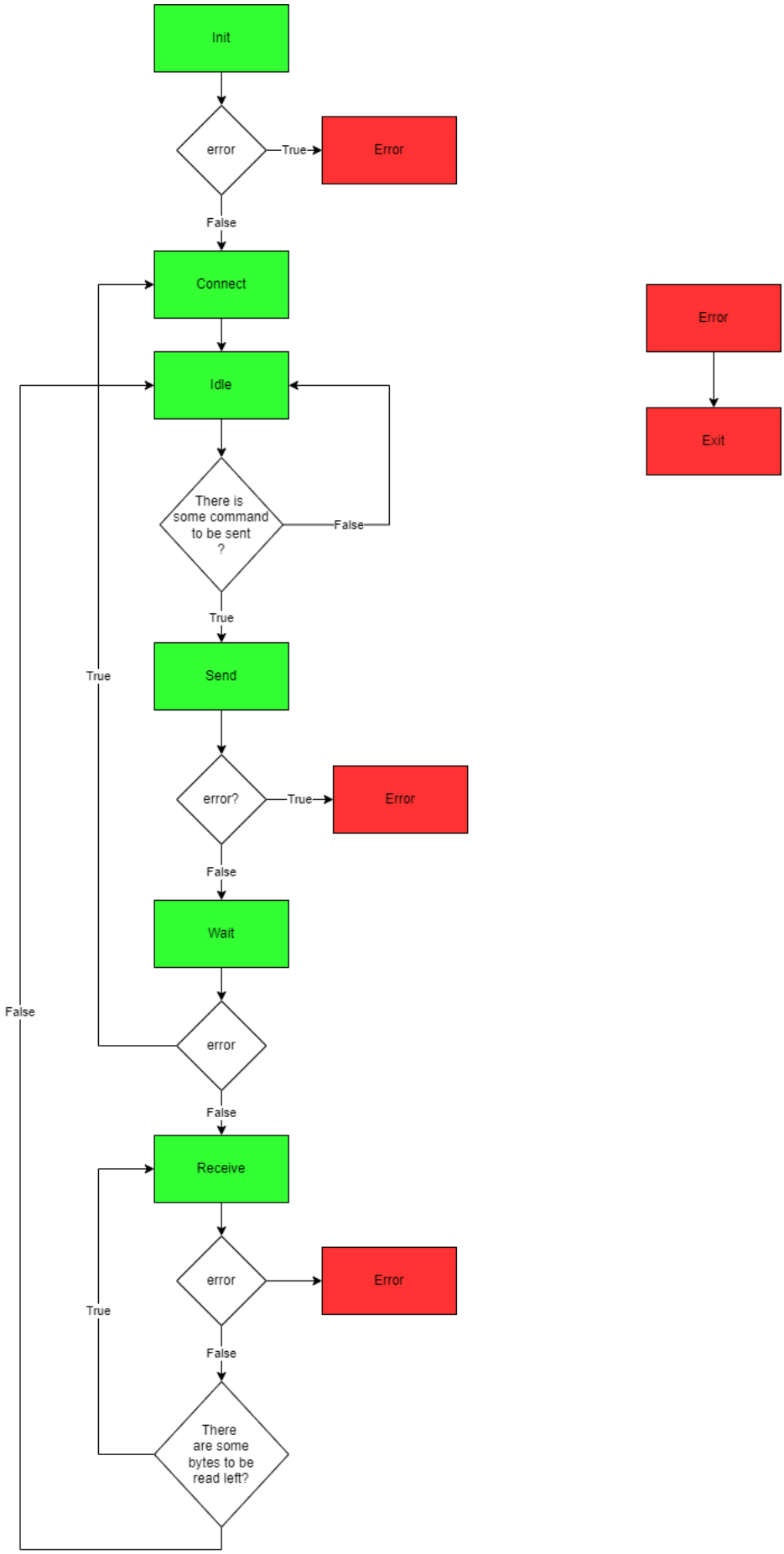


Figure 8: UART communication protocol between LabVIEW and PSoC

4 Real-time Autonomous Vehicle Control on the PSoC

4.1 Controlling the Actuators of the RC car

We used the PSOC to control the car and read the speed in real-time. It is connected to two actuators and six sensors. The two actuators are controlled at the same time, using the multi-tasking abilities of the PSOC chip. One in a DC motor, which defines the speed of the car, and the other is a servo-motor to control the direction of the car. They are both controlled by a PWM signal of same frequency and same duty cycle.

To determine the characteristics of this PWM signal, we used an oscilloscope that we hooked up in parallel while controlling the RC car with the remote control. This connection allowed us to know exactly which signal to input the controller of the RC car. We can see it on Figure20 This manipulation also helped us understand that the controller of the RC car had an initial condition: both PWM signals had to be in neural position to be able to connect to the PSOC and the controller. This was naturally done by the remote control but it was not the case with our code. So to be able to connect the PSOC in a safe way, we had to code a little initial condition on the PWM.

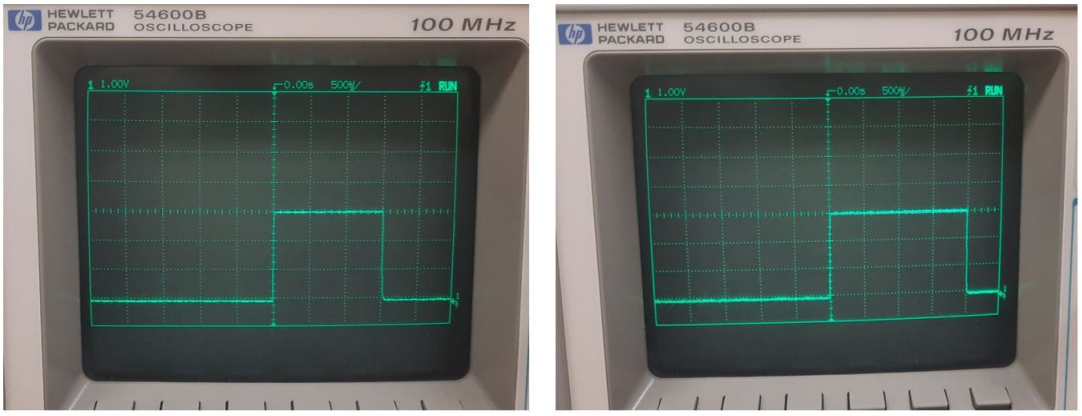


Figure 9: The output that gave our infrared sensor

We also noticed that the PWMs could suffer some interference with one another, basically, when we triggered the servomotor to turn, it sometimes entailed a quick acceleration of the DC motor. To prevent that effect, we had to make sure that the cables between the PSOC and the motors were of the say length, a multiple of the waive-length of the signal.

We coded the PSOC in order to have the ability to control the motors directly with the GUI via a UART communication or in an autonomous way with the interrupts linked to our sensors.

4.2 Measuring Velocity

To get the value of the speed, many options could have been used. The classic approach is often to use a Hall effect sensor. In our implementation we decided to explore the use of Infrared sensors. We had a large stock of these available, and decided to use this technology to count the rotations to explore the accuracy. To implement this, required applying a small strip of non-reflecting opaque tape on the axle of the car. The metal axle of the car by contrast is a reflective surface. For each rotation of this axle, a pulse is sent by the infrared sensor. Using a timer and an interrupt to track the number of state changes allowed for accurate measurement of trans-axle rotations. When the IR sensor logic switches from high to low, this state change and it triggers the interrupt. In the loop linked to the interrupt, we increment a value to count the rotations. After 1 second, the value of this counter is read and reset to zero to it in order to restart the loop. This counter is directly linearly proportional to the speed. The number of rotations was then scaled to convert the speed of the wheels in RPM. The scaling factor implemented was a factor of 3, diving the counter by 3 because the axle made 3 rotations for one rotation of the wheels. This approach accurately measured the speed of the car in real time, this value was used as the feedback value in order to implement PID control.

A crucial aspect of speed detection is the precision of the sensor: it indeed has to send one and only pulse for each rotation, otherwise, the speed may have an unacceptable margin of error. It is important to note that the sensor is not perfect, its output signal is not a perfect square signal. When we zoom on the output signal on an oscilloscope, note that there is actually a serrated curve from low to high shown on figure10. As the interrupt is sensitive to the change of state, this uneven signal was sometimes misinterpreted by the PSOC chip and the interrupt was triggered multiple times.

Figure 10: The output that gave our infrared sensor

To prevent that, we first tried to disable the interrupt for a few milliseconds after each trigger because it was during this time slot that the noise appeared and it could not prevent us from missing a rotation of the axle. This was effective, because in this time period we would not expect another rotation so soon, because we would never have the car at a max speed that would require sensing this RPM. This simple trick worked, however we did not like the fact that we had to put a delay in our code, or a new timer. It reduces the reactivity of the code base and it can downsize the benefits of having a real time chip. So we decided to keep a simple code and complicate the hardware instead. We ultimately used a Schmitt Trigger to "clean" the signal of the infrared sensor and virtually create a pure sq

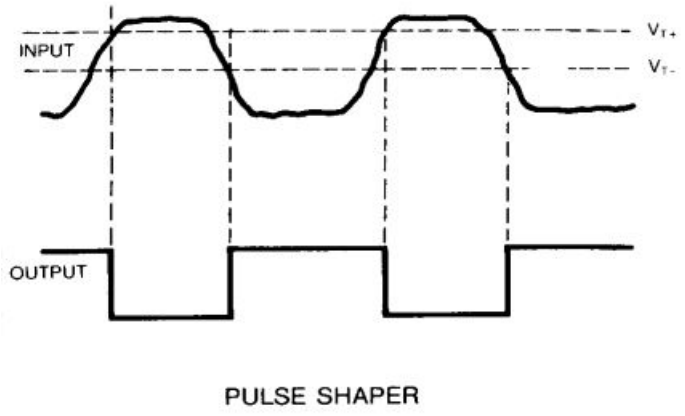


Figure 11: The effect of the inverter Schmitt Trigger

uare signal as shown on figure 11

To do so, we used a DSAP0027476 hex inverter Schmitt trigger chip. We then had to adapt our input voltage to have an appropriate value of V+ and V-. Indeed, the chip comes with values of V+ and V- that are proportional to its input voltage as shown on Figure 12. As our sensor was powered by a 3.3V from the PSOC, we have to estimate, for the values of V+ that we wanted, to calibrate input voltage. We found that to have a $V_+ = 0.75Vcc$, it was necessary to power the chip with $V_input = 1.57V$. We achieved this voltage using a potentiometer.

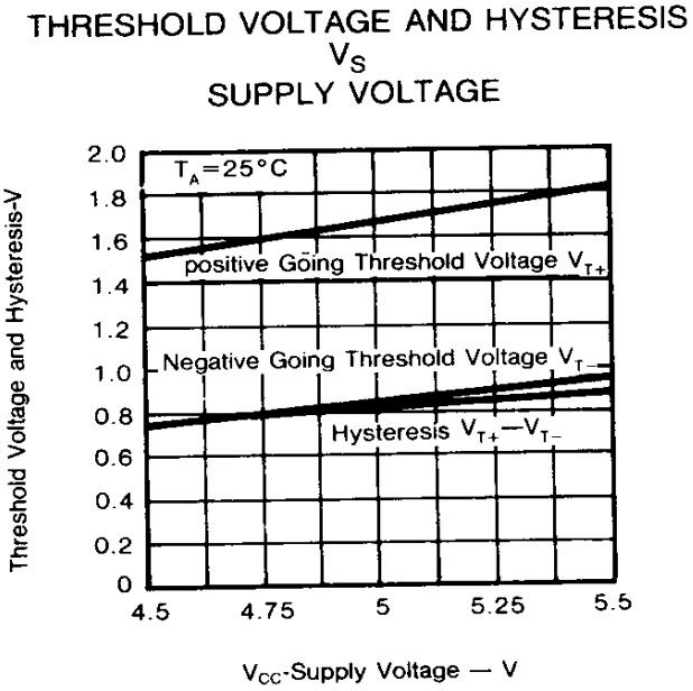


Figure 12: The values of the triggering voltages according to the data-sheet

With all these modifications, our output signal was perfectly squared and the value of the counter was reliable. The only downsize of this technique instead of having used an

accelerometer to get the speed of the car is that is does not have an accurate value as soon as the car slips on a driving surface and the wheels lose static friction with the contact plane. This however is an unavoidable issue with using a hall effect sensor as well. This potential issue however, was out of the scope of our project as the car is slowly mapping a room.

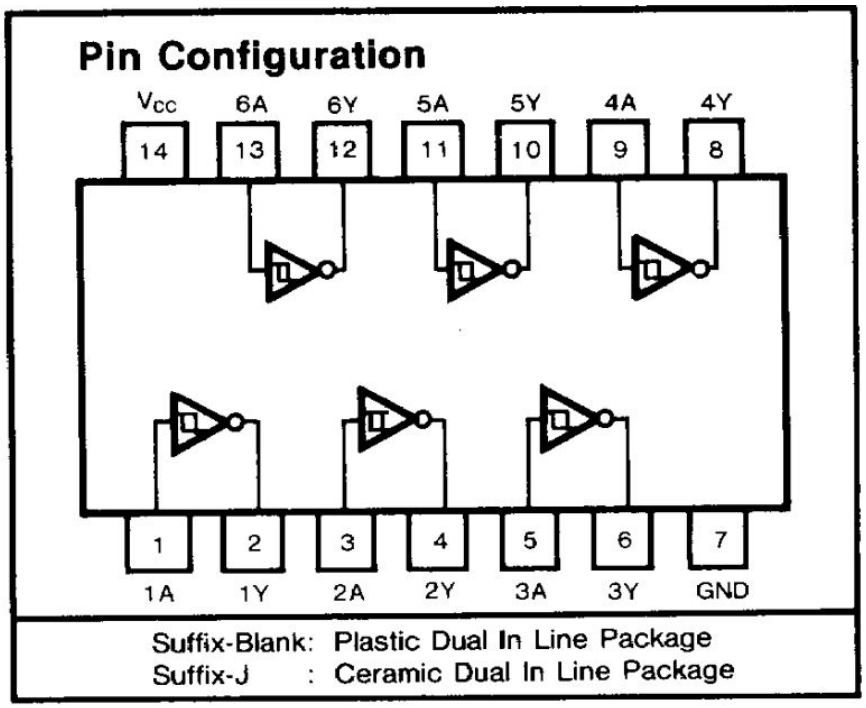


Figure 13: Electric diagram of the DSAP0027476 chip

4.3 Implementing Real-Time PID Control

We used the value of the speed to implement a closed loop PID control on the speed. The main asset of having the speed controlled instead of leaving an open loop is the ability to maintain the same speed regardless of the load on the car and of the angle of the road. Indeed, when the torque was too high for the car, the speed reduced inexorably to go a to a full stop of the car because the output value of the PWM was not evolving. But with a PID control, the PSOC knows in real time how fast the car is and can increase the duty cycle of the PWM if it detects that the car is not moving fast enough.

4.4 Controlling the direction with five infrared sensors

To control the direction of the car, we used five infrared sensor to track the position of the white line to follow. When the sensor was above the white line, the output was high and vice versa. So as soon as the sensor M detected the line, the car did not turn. Then, in all the other cases, the car implemented a certain angle depending on the position of the line as explained in Figure 23. We also implemented a variable "g" or "r" to remember the previous position of the line. For instance, if the line was between the sensor L1 and the sensor L2, nothing was detected by any sensors but because of this variable, we knew that

the line was between these two sensors. It allowed us to have a wider range of detection without adding more sensors. If a small variation is detected, a small angle is applied and if a big variation is detected, a big angle is applied. We also programmed the car to keep turning even if nothing was detected to rejoin the line even if the angle is too sharp. This is also allowed by our variable. This ultimately even allowed for error correcting if the car was to overshoot the line and lose visibility.

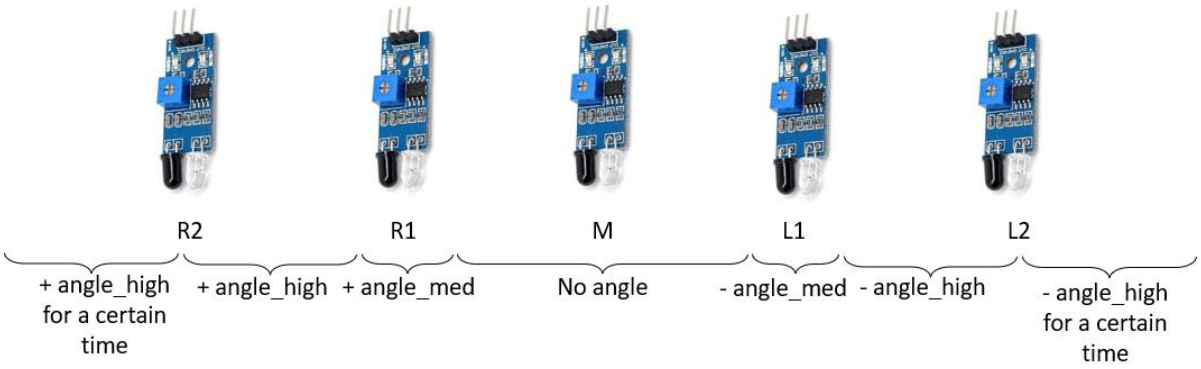


Figure 14: The logic of the command depending on the position of the white line

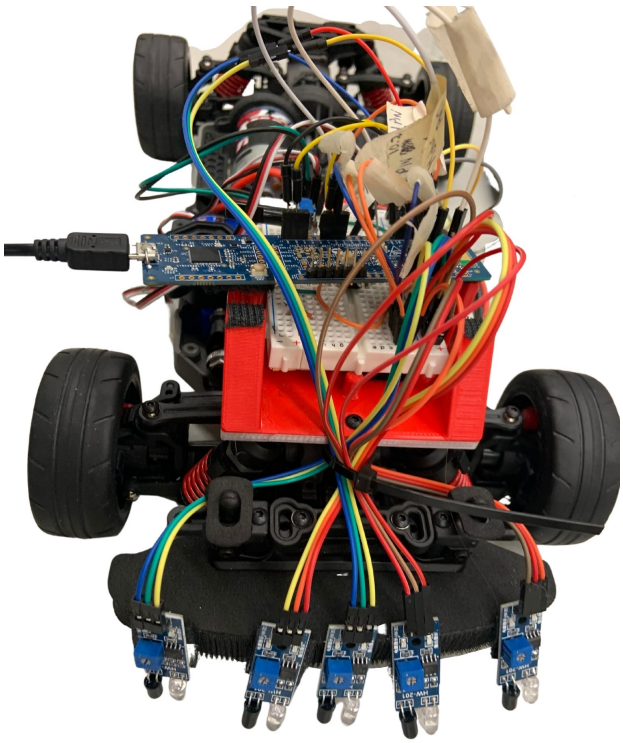


Figure 15: Early iteration IR sensor Configuration

5 LiDAR SLAM using Raspberry Pi

5.1 ROS2 Foxy & Multitasking

This section of the code base used ROS2 middleware and python code to interact with the LiDAR sensor using a 2GB raspberry pi4. A lightweight headless Linux distribution was installed on the raspberry pi4. The LiDAR data was sent back from the hardware LiDAR unit through the raspberry pi4 using a ROS topic via Bluetooth and a separate computer subscribed to the ROS LiDAR sensor data published on the network. This Linux computer also runs a ROS subscriber node locally to listen for messages from the Bluetooth transmitting raspberry pi4. The messages were received and RVIZ and Cartographer were spun up on the Linux PC that received transmission from the robot. The remote computer used cartographer SLAM to map the LiDAR sensor data and save the scanned environment into a map. The PC receiving the Map and the LiDAR sensor data shares the same wifi network as the raspberry pi4 allowing for an ssh connection to transmit data. This config file is used on the raspberry pi to connect the raspberry pi4 to the local network. The file name ***50-cloud-init.yaml*** was added on the microSD, and was required by the raspberry pi4 for a network connection via WiFi and added in the following directory.

```
/media/$USER/writable/etc/netplan
```

Once on the same network, ssh between the raspberry pi4 and the Linux machine was possible by obtaining the raspberry pi's IP, and using ifconfig and executing the following command on the remote PC.

```
$ ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
```

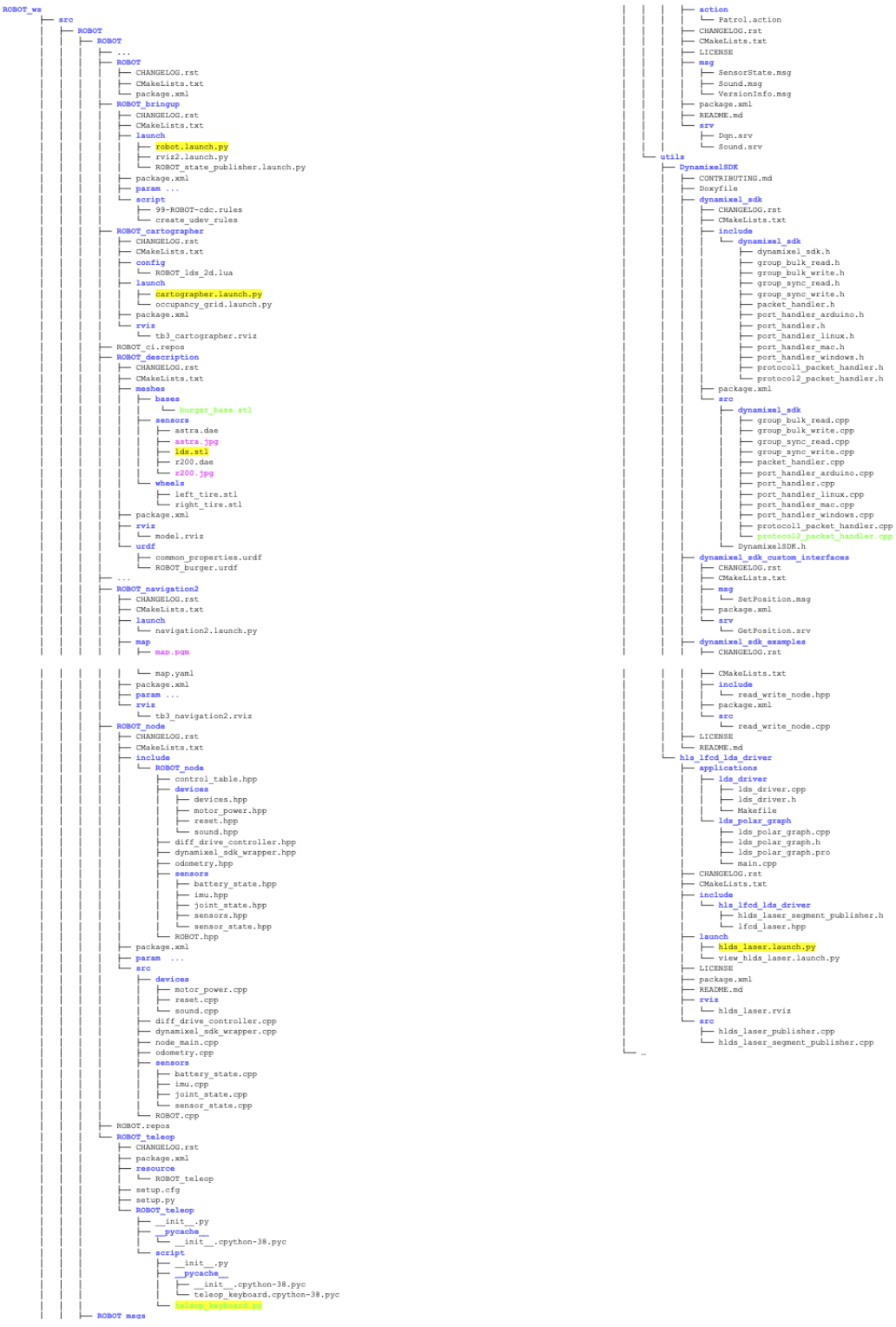
The ROS file structure is listed below. The sections focused on in this report will be the main launch files. Launch files can refer to other launch files in different directories, as visible in the code. To run the ROS packages requires a command of the following syntax after installing dependencies and building the ROS pacakges. The syntax of the launch run command is as as follows:

```
ros2 launch ROBOT_bringup robot.launch.py
ros2 launch ROBOT_cartographer cartographer.launch.py
```

The launch commands executed in the terminal bring up the respective publisher and subscriber nodes. While the robot.launch.py file is run via ssh in the raspberry pi, the cartographer.launch.py file is run on the PC to subscribe to the LiDAR data published via ssh.

5.2 ROS2 Workspace File Structure

ROS is a middleware where "nodes" can be written in python or C++, "nodes" actively publish and subscribe to data threads. The file-structure for the code base for for our SLAM impolememted publisher and subscriber nodes executed with launch files. The code base file tree is as follows, highlighted files will be discussed. **A GitHub source link will be included in the appendix.**


ROBOT_ws
src
ROBOT
ROBOT
...
ROBOT
CHANGELOG.rst
CMakeLists.txt
package.xml
ROBOT_bringup
CHANGELOG.rst
CMakeLists.txt
launch
robot.launch.py
rviz2.launch.py
ROBOT_state_publisher.launch.py
package.xml
param ...
script
99-ROBOT-cdc.rules
create_udev_rules
ROBOT_cartographer
CHANGELOG.rst
CMakeLists.txt
config
ROBOT_lds_2d.lua
launch
cartographer.launch.py
occupancy_grid.launch.py
package.xml
rviz
tb3_cartographer.rviz
ROBOT_ci.repos
ROBOT_description
CHANGELOG.rst
CMakeLists.txt
meshes
bases
burger_base.stl
sensors
astra.dae
astra.jpg
lds.stl
r200.dae
r200.jpg
wheels
left_tire.stl
right_tire.stl
package.xml
rviz
model.rviz
urdf
common_properties.urdf
ROBOT_burger.urdf
...
ROBOT_navigation2
CHANGELOG.rst
CMakeLists.txt
launch
navigation2.launch.py
map
map.pgm
map.yaml
package.xml
param ...
rviz
tb3_navigation2.rviz
ROBOT_node
CHANGELOG.rst
CMakeLists.txt
include
ROBOT_node
control_table.hpp
devices
devices.hpp
motor_power.hpp
reset.hpp
sound.hpp
diff_drive_controller.hpp
dynamixel_sdk_wrapper.hpp
odometry.hpp
sensors
battery_state.hpp
imu.hpp
joint_state.hpp
sensors.hpp
sensor_state.hpp
ROBOT.hpp
package.xml
param ...
src
devices
motor_power.cpp
reset.cpp
sound.cpp
diff_drive_controller.cpp
dynamixel_sdk_wrapper.cpp
node_main.cpp
odometry.cpp
sensors
battery_state.cpp
imu.cpp
joint_state.cpp
sensor_state.cpp
ROBOT.cpp
ROBOT.repos
ROBOT_teleop
CHANGELOG.rst
package.xml
resource
ROBOT_teleop
setup.cfg
setup.py
ROBOT_teleop
__init__.py
__pycache__
__init__.cpython-38.pyc
script
__init__.py
__pycache__
__init__.cpython-38.pyc
teleop_keyboard.cpython-38.pyc
teleop_keyboard.py
ROBOT msgs
action
Patrol.action
CHANGELOG.rst
CMakeLists.txt
LICENSE
msg
SensorState.msg
Sound.msg
VersionInfo.msg
package.xml
README.md
srv
Dqn.srv
Sound.srv
utils
DynamixelSDK
CONTRIBUTING.md
Doxyfile
dynamixel_sdk
CHANGELOG.rst
CMakeLists.txt
include
dynamixel_sdk
dynamixel_sdk.h
group_bulk_read.h
group_bulk_write.h
group_sync_read.h
group_sync_write.h
packet_handler.h
port_handler_arduino.h
port_handler.h
port_handler_linux.h
port_handler_mac.h
port_handler_windows.h
protocol1_packet_handler.h
protocol2_packet_handler.h
package.xml
src
dynamixel_sdk
group_bulk_read.cpp
group_bulk_write.cpp
group_sync_read.cpp
group_sync_write.cpp
packet_handler.cpp
port_handler_arduino.cpp
port_handler.cpp
port_handler_linux.cpp
port_handler_mac.cpp
port_handler_windows.cpp
protocol1_packet_handler.cpp
protocol2_packet_handler.cpp
DynamixelSDK.h
dynamixel_sdk_custom_interfaces
CHANGELOG.rst
CMakeLists.txt
msg
SetPosition.msg
package.xml
srv
GetPosition.srv
dynamixel_sdk_examples
CHANGELOG.rst
CMakeLists.txt
include
read_write_node.hpp
package.xml
src
read_write_node.cpp
LICENSE
README.md
hls_lfcd_lds_driver
applications
lds_driver
lds_driver.cpp
lds_driver.h
Makefile
lds_polar_graph
lds_polar_graph.cpp
lds_polar_graph.h
lds_polar_graph.pro
main.cpp
CHANGELOG.rst
CMakeLists.txt
include
hls_lfcd_lds_driver
hlds_laser_segment_publisher.h
lfcd_laser.hpp
launch
hlds_laser.launch.py
view_hlds_laser.launch.py
LICENSE
package.xml
README.md
rviz
hlds_laser.rviz
src
hlds_laser_publisher.cpp
hlds_laser_segment_publisher.cpp

5.2.1 ld08.launch.py

link to LiDAR driver source **here** The driver used for the LiDAR sensor was installed into the ROS2 workspace on the raspberry pi4 using the following command.

```
git clone -b ros2-devel https://github.com/ROBOTIS-GIT/ld08_driver.git
```

This launch file is nested in the main launch files, and brings up the driver to interact with the LiDAR hardware sensor.

```
#!/usr/bin/env python3

import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.actions import LogInfo
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node


def generate_launch_description():

    return LaunchDescription([
        Node(
            package='ld08_driver',
            executable='ld08_driver',
            name='ld08_driver',
            output='screen'),
    ])
```

5.2.2 ROBOT_bringup_robot.launch.py

This is the main launch file to bring up the LiDAR and the respective publishing nodes. There are additional packages launched that are artifacts from testing the LiDAR sensor with a standard 2 wheel wheelbase early on to get the software stack up and running. The command to launch the launch file is as follows:

```
ros2 launch ROBOT_bringup robot.launch.py
```

The launch file and respective packages are shown here with comments

```
#!/usr/bin/env python3

import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch.substitutions import ThisLaunchFileDir
```

```
from launch_ros.actions import Node


def generate_launch_description():
    TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']
    LDS_MODEL = os.environ['LDS_MODEL']
    LDS_LAUNCH_FILE = '/hlds_laser.launch.py'

    usb_port = LaunchConfiguration('usb_port', default='/dev/ttyACM0')

    tb3_param_dir = LaunchConfiguration(
        'tb3_param_dir',
        default=os.path.join(
            get_package_share_directory('turtlebot3_bringup'),
            'param',
            TURTLEBOT3_MODEL + '.yaml'))

    # incorporating the lidar module driver files
    if LDS_MODEL == 'LDS-01':
        lidar_pkg_dir = LaunchConfiguration(
            'lidar_pkg_dir',
            default=os.path.join(get_package_share_directory('hls_lfcd_lds_driver'), 'launch'))
    elif LDS_MODEL == 'LDS-02':
        lidar_pkg_dir = LaunchConfiguration(
            'lidar_pkg_dir',
            default=os.path.join(get_package_share_directory('ld08_driver'), 'launch'))
        LDS_LAUNCH_FILE = '/ld08.launch.py'
    else:
        lidar_pkg_dir = LaunchConfiguration(
            'lidar_pkg_dir',
            default=os.path.join(get_package_share_directory('hls_lfcd_lds_driver'), 'launch'))

    use_sim_time = LaunchConfiguration('use_sim_time', default='false')

    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value=use_sim_time,
            description='Use simulation (Gazebo) clock if true'),

        DeclareLaunchArgument(
            'usb_port',
            default_value=usb_port,
            description='Connected USB port with OpenCR'),

        DeclareLaunchArgument(
            'tb3_param_dir',
            default_value=tb3_param_dir,
            description='Full path to turtlebot3 parameter file to load'),

        # continually publish state information to allow the subscriber node access
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
```

```
                [ThisLaunchFileDir(), '/turtlebot3_state_publisher.launch.py']),
            launch_arguments={'use_sim_time': use_sim_time}.items(),
        ),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([lidar_pkg_dir, LDS_LAUNCH_FILE]),
            launch_arguments={'port': '/dev/ttyUSB0', 'frame_id': 'base_scan'}.items()
    ,
        ),

        Node(
            package='turtlebot3_node',
            executable='turtlebot3_ros',
            parameters=[tb3_param_dir],
            arguments=['-i', usb_port],
            output='screen'),
    ])
```

5.2.3 ROBOT_state_publisher.launch.py

This file just continually broadcasts the state of all the urdf parameters configured for the robot. This allows subscriber nodes to see published states for the parameters in the urdf files. This is how the subscriber gets position or sensor updates and sensor information.

```

import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare


def generate_launch_description():
    TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']

    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    urdf_file_name = 'turtlebot3_' + TURTLEBOT3_MODEL + '.urdf'

    print("urdf_file_name : {}".format(urdf_file_name))

    urdf = os.path.join(
        get_package_share_directory('turtlebot3_description'),
        'urdf',
        urdf_file_name)

    # Major refactor of the robot_state_publisher
    # Reference page: https://github.com/ros2/demos/pull/426
    with open(urdf, 'r') as infp:
        robot_desc = infp.read()

    rsp_params = {'robot_description': robot_desc}
```

```
31
32      # print (robot_desc) # Printing urdf information.
33
34      return LaunchDescription([
35          DeclareLaunchArgument(
36              'use_sim_time',
37              default_value='false',
38              description='Use simulation (Gazebo) clock if true'),
39          Node(
40              package='robot_state_publisher',
41              executable='robot_state_publisher',
42              output='screen',
43              parameters=[rsp_params, {'use_sim_time': use_sim_time}])
44      ])
```

5.2.4 ROBOT_cartographer cartographer.launch.py

This is the launch file to run the subscriber node for SLAM locally on the PC and subscribe to the LiDAR data through the respective publishing nodes. The command to launch the launch file is as follows:

```
1 ros2 launch ROBOT_cartographer cartographer.launch.py
```

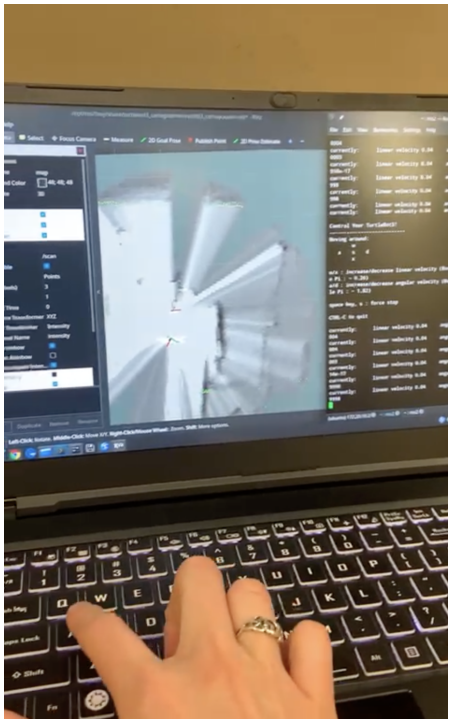


Figure 17: Simultaneous Localization and Mapping SLAM outcomes with low resolution LiDAR sensor in Google Cartographer Package

The launch file and respective packages are shown here with comments

```
1
```

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.conditions import IfCondition
from launch.actions import DeclareLaunchArgument
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir


# import relevant packages needed by cartographer
def generate_launch_description():
    use_rviz = LaunchConfiguration('use_rviz', default='true')
    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    turtlebot3_cartographer_prefix = get_package_share_directory('turtlebot3_cartographer')
    cartographer_config_dir = LaunchConfiguration('cartographer_config_dir', default=os.path.join(
                                                turtlebot3_cartographer_prefix, 'config'))
    configuration_basename = LaunchConfiguration('configuration_basename',
                                                default='turtlebot3_lds_2d.lua')

    resolution = LaunchConfiguration('resolution', default='0.05')
    publish_period_sec = LaunchConfiguration('publish_period_sec', default='1.0')

    rviz_config_dir = os.path.join(get_package_share_directory('turtlebot3_cartographer'),
                                   'rviz', 'tb3_cartographer.rviz')

    return LaunchDescription([
        DeclareLaunchArgument(
            'cartographer_config_dir',
            default_value=cartographer_config_dir,
            description='Full path to config file to load'),
        DeclareLaunchArgument(
            'configuration_basename',
            default_value=configuration_basename,
            description='Name of lua file for cartographer'),
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use simulation (Gazebo) clock if true'),

        Node(
            package='cartographer_ros',
            executable='cartographer_node',
            name='cartographer_node',
            output='screen',
            parameters=[{'use_sim_time': use_sim_time}],
            arguments=['-configuration_directory', cartographer_config_dir,
                       '-configuration_basename', configuration_basename]),

        DeclareLaunchArgument(
```

```
            'resolution',
            default_value=resolution,
            description='Resolution of a grid cell in the published occupancy grid'),

        DeclareLaunchArgument(
            'publish_period_sec',
            default_value=publish_period_sec,
            description='OccupancyGrid publishing period'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([ThisLaunchFileDir(), '/occupancy_grid.launch.py']),
            launch_arguments={'use_sim_time': use_sim_time, 'resolution': resolution,
                              'publish_period_sec': publish_period_sec}.items(),
        ),

        # this launch file automatically will bring up rviz, rviz will turn on running the cartographer package
        Node(
            package='rviz2',
            executable='rviz2',
            name='rviz2',
            arguments=['-d', rviz_config_dir],
            parameters=[{'use_sim_time': use_sim_time}],
            condition=IfCondition(use_rviz),
            output='screen'),
    ])
```

6 Issues

6.1 CAD Chassis Cage Not Completed

It was hard to get these parts 3D printed due to the last minute request to get them printed. if given the opportunity to go back and redo the project, would definitely get these printed out earlier on.

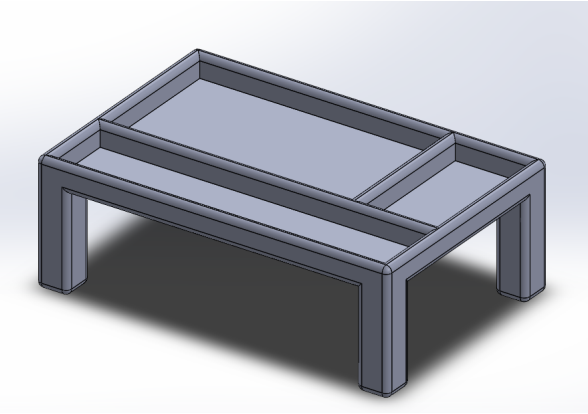


Figure 18: The top cover meant to hold the PSOC and Raspberry Pi

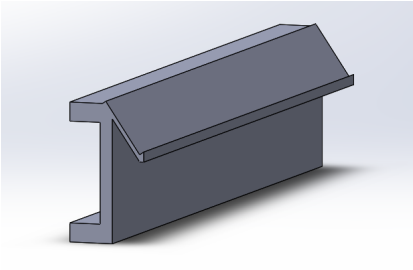


Figure 19: The front bumber cover meant to hold the five front IR sensors

6.2 Low Resolution LiDAR Sensor

If given the opportunity to go back in time, it would be nice to aquire a 3D higher resolution LiDAR scanner. The LiDAR sensor used in conjunction with ROS2 and google's Cartographer package, was a single line 360 degree scanner 2D LiDAR sensor from a Robotis, originally used on a turtlebot3 model. This LiDAR sensor is supported with a ROS2 software driver allowing you to interact and develop ROS code to process the LiDAR data. We did so using a google Cartographer package. The outcomes were not ideal because of the low resolution and scan speed. The LiDAR point cloud shows the highest accuracy for slow passes, with higher resolution sensors with a larger field of view higher speeds can be achieved, however these LiDAR scanners are prohibitively expensive. The model we acquired for this project was around the $200-$300 range alone.

Figure 20: LDS-01 LiDAR sensor, displayed in the opening section. Single line LiDAR scanner, off the shelf cost $200-$300 range

However as you can see from the images collected our outcomes were not sufficient resolution for higher speed obstacle detection or object classification. This scanner could generate slightly higher resolution maps if allowed multiple incredibly slow passes of a room over the course of many hours, but the resolution leads to poor error correcting on secondary passes. These sample images shown below were after about 20 minutes of scanning. The Turtlebot was able to partially map the lab room.

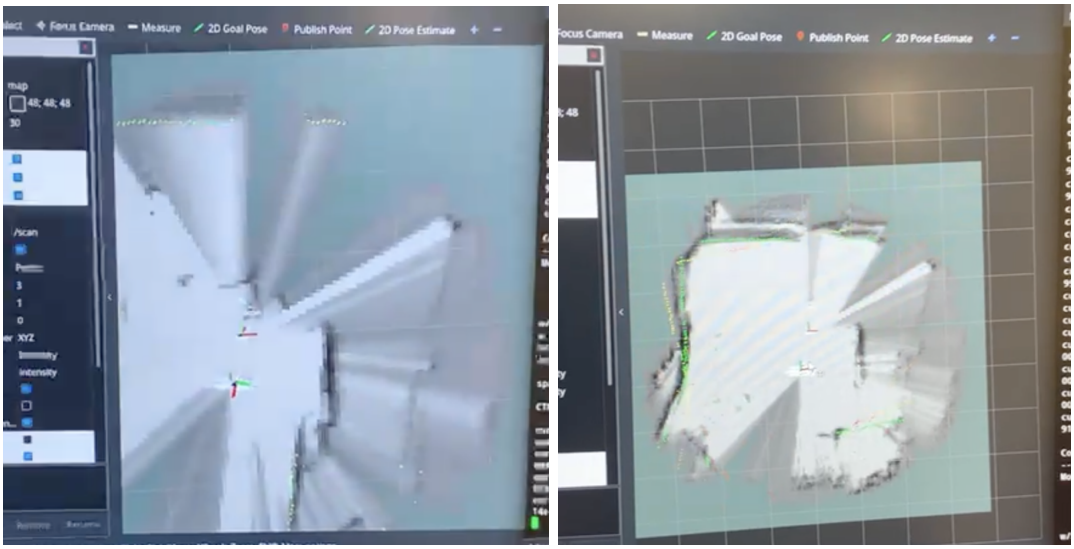


Figure 21: Simultaneous Localization and Mapping SLAM outcomes with low resolution LiDAR sensor in Google Cartographer Package. This scan shown is the same room, the left image an earlier time step 15 minutes prior with less scan lines accumulated, the right is the later time step

The Black sections show the SLAM detected obstacles saved as a wall. Note the fuzzy offset comes from overlaying points and a degree of error in calculating the correct offset for feature points on the map. This offset is exacerbated at lower resolutions when the LiDAR struggles to identify unique features to provide to the SLAM package to error correct, and it relies heavily on the wheel odometetry/rpm to offset the scanned point clouds.
With a higher resolution sensor the Cartographer package produces much better results, an example of a Cartographer map created by a better resolution sensor where object/map-feature classification shows a significantly higher degree of accuracy is shown below. The limiting factor for a better LiDAR SLAM implementation was cost.

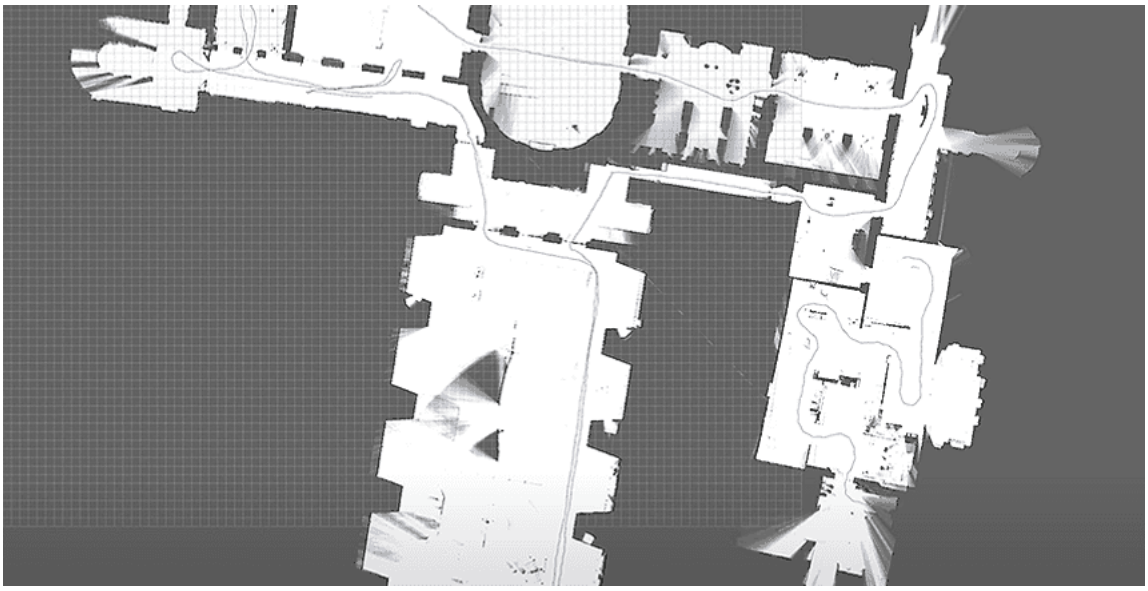


Figure 22: Cartographer python library with a higher cost and higher resolution LiDAR sensor

6.3 Trying to implement BLE wireless connection via PSOC6

If given the opportunity to go back it would be a better use of time to focus on only implementing BLE on the Raspberry Pi and starting sooner on this task. BLE on the PSoC was a real challenge for us, as the wireless feature was what we expected the robot to perform.
After buying a bluetooth BLE dongle and using a wireless chip and using a template of LabVIEW BLE communication, we were able to use the **Find me** bluetooth basic example of implementation on the PSoC and perform the three basic tasks that this example allowed from the GUI. However, after a lot of effort, we did not succeed in creating our own tailored tasks. The goals were sending our own specific string commands, and receiving status updates from the robot. This is due to two different reasons:

- It is very complex to send instructions from both ends of the system
- We had a hard time defining and configuring a new BLE attribute to communicate through.

We think with more time and more efforts it would have been possible, but due to lack of resource, and for the sake of the demonstration, we believed it was better to keep the wired connection and focus on some more important aspect of the project.

6.4 IR Sensors - Close range Accuracy

Integrating the IR sensors with the LiDAR could have been a more robust to keep the Robot "on track". The IR sensor design worked very well for this project, however using them for line detection posed some constraints, namely the position of the IR sensors were a limiting factor for the car speed. The IR sensors on the front bumper could not "detect ahead". Their response rate was incredibly quick, but they can only sense the line while

directly over it. While the ideal implementation of this experimental car intended to use the LiDAR in conjunction for navigation purposes, the resolution of this LiDAR unit was not feasible for low error object detection at speed. With a higher resolution LiDAR unit, the IR sensors prove to be a high accuracy low-cost redundant system to detect the car's position relative to a lane marker. In a racing situation the success of this test is of high value. While Camera and LiDAR systems used for lane detection are common, low cost high accuracy sensors that provide accurate odometetry and lane sensing provide a mission critical source for error correction to help greatly improve positioning accuracy and pose estimation. Future concerns and experimentation involve testing how robust the system is in low light conditions. Our limited testing showed no differences, but this could be an area of future exploration.

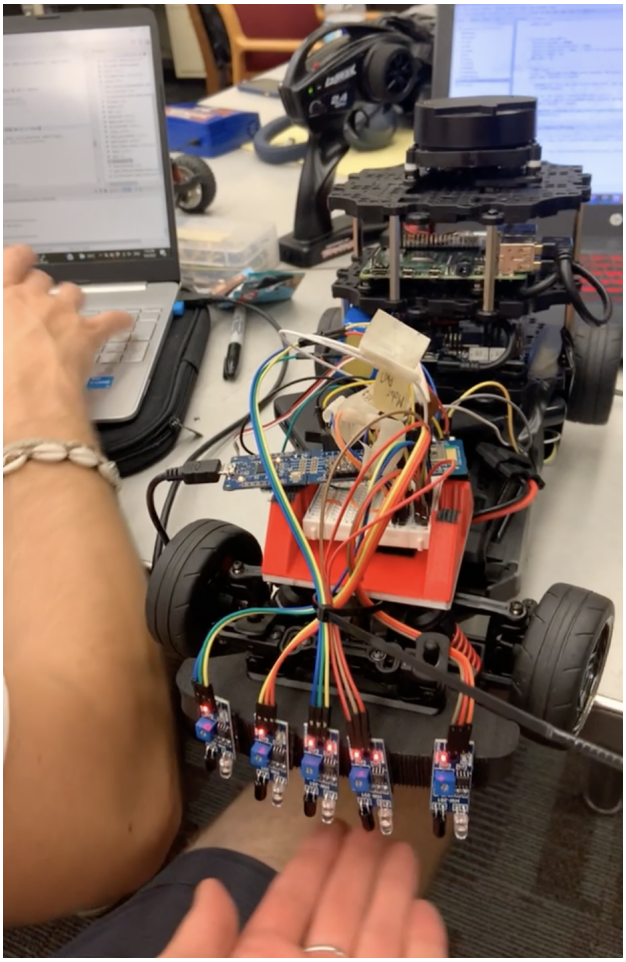


Figure 23: Experimental IR sensor Configuration, Note 2 LEDs activated by hand under central IR sensors

7 Appendix

7.1 PSoC code for controlling the RC car

```
/*******************************************************************************
* File Name:   main.c
*
* Description: This is the source code for the PSoC 6 MCU: Hello World Example
*              for ModusToolbox.
*
* Related Document: See README.md
*
*
********************************************************************************
* Copyright 2019-2021, Cypress Semiconductor Corporation (an Infineon company) or
* an affiliate of Cypress Semiconductor Corporation.  All rights reserved.
*
* This software, including source code, documentation and related
* materials ("Software") is owned by Cypress Semiconductor Corporation
* or one of its affiliates ("Cypress") and is protected by and subject to
* worldwide patent protection (United States and foreign),
* United States copyright laws and international treaty provisions.
* Therefore, you may use this Software only as provided in the license
* agreement accompanying the software package from which you
* obtained this Software ("EULA").
* If no EULA applies, Cypress hereby grants you a personal, non-exclusive,
* non-transferable license to copy, modify, and compile the Software
* source code solely for use in connection with Cypress's
* integrated circuit products.  Any reproduction, modification, translation,
* compilation, or representation of this Software except as specified
* above is prohibited without the express written permission of Cypress.
*
* Disclaimer: THIS SOFTWARE IS PROVIDED AS-IS, WITH NO WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT, IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress
* reserves the right to make changes to the Software without notice. Cypress
* does not assume any liability arising out of the application or use of the
* Software or any product or circuit described in the Software. Cypress does
* not authorize its products for use in any products where a malfunction or
* failure of the Cypress product may reasonably be expected to result in
* significant property damage, injury or death ("High Risk Product"). By
* including Cypress's product in a High Risk Product, the manufacturer
* of such system or application assumes all risk of such use and in doing
* so agrees to indemnify Cypress against all liability.
*******************************************************************************/

#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "cy_retarget_io.h"


/*******************************************************************************
* Macros
*******************************************************************************/
```

```
52
53 #define LED_BLINK_TIMER_CLOCK_HZ          (10000) //LED blink timer clock value in Hz
54 #define LED_BLINK_TIMER_PERIOD            (4999) //LED blink timer period value
55
56 #define GPIO_INTERRUPT_PRIORITY (7u)
57
58
59 /*******************************************************************************
60 * Function Prototypes
61 *******************************************************************************/
62 void timer_init(void);
63 static void isr_timer(void *callback_arg, cyhal_timer_event_t event);
64 static void gpio_interrupt_handler(void *handler_arg, cyhal_gpio_event_t event);
65
66
67 /*******************************************************************************
68 * Global Variables
69 *******************************************************************************/
70 bool timer_interrupt_flag = false;
71 bool led_blink_active_flag = true;
72
73 volatile bool gpio_intr_flag = false;
74
75 /* Variable for storing character read from terminal */
76 uint8_t uart_read_value;
77
78 /* Timer object used for blinking the LED */
79 cyhal_timer_t led_blink_timer;
80
81
82
83 /* RC car variables */
84 float speed=0;
85 uint32_t count = 0;
86 bool IR_pause = false;
87
88 float period = LED_BLINK_TIMER_PERIOD;
89
90
91 /*******************************************************************************
92 * Function Name: main
93 ********************************************************************************
94 * Summary:
95 * This is the main function for CM4 CPU. It sets up a timer to trigger a
96 * periodic interrupt. The main while loop checks for the status of a flag set
97 * by the interrupt and toggles an LED at 1Hz to create an LED blinky. The
98 * while loop also checks whether the 'Enter' key was pressed and
99 * stops/restarts LED blinking.
100 *
101 * Parameters:
102 *  none
103 *
104 * Return:
105 *  int
106 *
107 *******************************************************************************/
```

```
int main(void)
{
    cy_rslt_t result;

    /* PWM object */
    cyhal_pwm_t pwm_servo_control;
    cyhal_pwm_t pwm_motor_control;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

   // Init GPIOs and interrupt

   result = cyhal_gpio_init(Pin_IR, CYHAL_GPIO_DIR_INPUT,
                            CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);

    cyhal_gpio_register_callback(Pin_IR, gpio_interrupt_handler, NULL);
    cyhal_gpio_enable_event(Pin_IR, CYHAL_GPIO_IRQ_FALL, GPIO_INTERRUPT_PRIORITY, true
    );

    result = cyhal_gpio_init(Pin_IR_D1, CYHAL_GPIO_DIR_INPUT,
                         CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);
    result = cyhal_gpio_init(Pin_IR_D2, CYHAL_GPIO_DIR_INPUT,
                         CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);
    result = cyhal_gpio_init(Pin_IR_G1, CYHAL_GPIO_DIR_INPUT,
                         CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);
    result = cyhal_gpio_init(Pin_IR_G2, CYHAL_GPIO_DIR_INPUT,
                         CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                                 CY_RETARGET_IO_BAUDRATE);

    /* retarget-io init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize the User LED */
    result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
                             CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

    /* GPIO init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
```

```
163     }
164
165     /* Initialize timer to toggle the LED */
166        timer_init();
167
168    /* Initialize the PWM */
169         uint32_t period = 11000;
170         uint32_t duty_cycle_neutral = 1550;
171         uint32_t speed_init = 1620;
172
173
174       //PWM servo
175       result = cyhal_pwm_init(&pwm_servo_control, Pin_servo, NULL);
176       if(CY_RSLT_SUCCESS != result)
177       {
178           printf("API cyhal_pwm_init failed with error code: %lu\r\n", (unsigned long)
    result);
179           CY_ASSERT(false);
180       }
181       /* Set the PWM output frequency and duty cycle */
182       result = cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral);
183       if(CY_RSLT_SUCCESS != result)
184       {
185           printf("API cyhal_pwm_set_duty_cycle failed with error code: %lu\r\n", (
    unsigned long) result);
186           CY_ASSERT(false);
187       }
188       /* Start the PWM */
189       result = cyhal_pwm_start(&pwm_servo_control);
190       if(CY_RSLT_SUCCESS != result)
191       {
192           printf("API cyhal_pwm_start failed with error code: %lu\r\n", (unsigned long)
     result);
193           CY_ASSERT(false);
194       }
195
196
197
198       // PWM Motor
199
200       result = cyhal_pwm_init(&pwm_motor_control, Pin_motor, NULL);
201       if(CY_RSLT_SUCCESS != result)
202       {
203           printf("API cyhal_pwm_init failed with error code: %lu\r\n", (unsigned long)
    result);
204           CY_ASSERT(false);
205       }
206       /* Set the PWM output frequency and duty cycle */
207       result = cyhal_pwm_set_period(&pwm_motor_control, period, duty_cycle_neutral);
208       if(CY_RSLT_SUCCESS != result)
209       {
210           printf("API cyhal_pwm_set_duty_cycle failed with error code: %lu\r\n", (
    unsigned long) result);
211           CY_ASSERT(false);
212       }
213       /* Start the PWM */
```

```
    result = cyhal_pwm_start(&pwm_motor_control);
    if(CY_RSLT_SUCCESS != result)
    {
        printf("API cyhal_pwm_start failed with error code: %lu\r\n", (unsigned long)
 result);
        CY_ASSERT(false);
    }


printf("PWM started successfully. Entering the sleep mode...\r\n");


uint8_t ln_readable;
char text[10];
int x;
uint32_t time_count = 0;
uint32_t dc = 0;

bool autonomous = false;

 bool G1, G2, D1 , D2;

 uint32_t angle_low = 150, angle_med = 300, angle_high = 450;
 uint32_t t_saved=0;

 for (;;)
 {
     /* Check if 'Enter' key was pressed */
     if (cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1)
          == CY_RSLT_SUCCESS)
     {
      switch(uart_read_value)
         {
         case 'T':
            if (led_blink_active_flag)
                                {
                                    cyhal_timer_stop(&led_blink_timer);

                                    printf("TLED blinking paused \r\n");
                                }
                                else /* Resume LED blinking by starting the timer */
                                {
                                    cyhal_timer_start(&led_blink_timer);

                                    printf("TLED blinking resumed\r\n");
                                }
            led_blink_active_flag ^= 1;
            break;
         case 'A':
            printf("ACar launched\r\n");
            dc = 0;
            while (dc + duty_cycle_neutral<speed_init)
            {
               dc = dc + 10;
               cyhal_pwm_set_period(&pwm_motor_control, period, dc+
 duty_cycle_neutral);
```

```
268                     CyDelay(100);
269                 }
270                 break;
271 
272             case 'B':
273                 printf("BCar stopped\r\n");
274                 cyhal_pwm_set_period(&pwm_motor_control, period, duty_cycle_neutral);
275 
276                 break;
277 
278             case 'U':
279                 printf("USpeed ++\r\n");
280                 speed_init = speed_init + 5;
281                 cyhal_pwm_set_period(&pwm_motor_control, period, speed_init);
282                 break;
283 
284             case 'D':
285                 printf("BReceived the B Command\r\n");
286                 speed_init = speed_init - 5;
287                 cyhal_pwm_set_period(&pwm_motor_control, period, speed_init);
288                 break;
289 
290             case 'O':
291                 printf("OTurn right\r\n");
292                 cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral
    +200);
293                 break;
294             case 'F':
295                 printf("FTurn left\r\n");
296                 cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral
    -200);
297                 break;
298             case 'I':
299                  printf("IPSOC\r\n");
300                 break;
301             case 'Y':
302                 time_count = time_count + 1;
303                  printf("Y %f\r\n", speed);
304 //                  printf("YTimer = %lu\r\n", cyhal_timer_read(&led_blink_timer));
305                 break;
306 
307             case 'V':
308                 cyhal_system_delay_ms(5);
309 
310                 ln_readable = cyhal_uart_readable(&cy_retarget_io_uart_obj);
311 
312                 for (int i=0; i<=ln_readable; i++)
313                 {
314                     cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1);
315                     text[i]=(char)uart_read_value;
316                 }
317                 x = atoi(text);
318                 printf("VSteering %d end\r\n", x);
319                 memset(text, 0, sizeof(text));
320 
321                 cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral+x);
```

```
322                 break;
323 
324             case 'X':
325                 cyhal_system_delay_ms(5);
326 
327                 ln_readable = cyhal_uart_readable(&cy_retarget_io_uart_obj);
328 
329                 for (int i=0; i<=ln_readable; i++)
330                 {
331                    cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1);
332                    text[i]=(char)uart_read_value;
333 //                    text[i]=uart_read_value;
334                 }
335                 x = atoi(text);
336                 printf("XThrottle %d\r\n", x);
337                 memset(text, 0, sizeof(text));
338 
339                 cyhal_pwm_set_period(&pwm_motor_control, period, duty_cycle_neutral+x);
340                 break;
341 
342             case 'G':
343                 printf("GAutonomous on\r\n");
344                 autonomous = true;
345                 break;
346 
347 
348             case 'H':
349                 printf("HAutonomous off\r\n");
350                 autonomous = false;
351 
352                 break;
353 
354             default:
355                 break;
356             }
357 
358       }
359 
360         if(autonomous == true && cyhal_timer_read(&led_blink_timer)-t_saved>1000)
361         {
362          G1 = cyhal_gpio_read(Pin_IR_G1);
363          G2 = cyhal_gpio_read(Pin_IR_G2);
364          D1 = cyhal_gpio_read(Pin_IR_D1);
365          D2 = cyhal_gpio_read(Pin_IR_D2);
366 
367          G1 = !G1;
368          G2 = !G2;
369          D1 = !D1;
370          D2 = !D2;
371 
372          if (G2 == false && G1 == true && D1 == true && D2 == false)
373          {
374             cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral);
375          }
376 
377          else if (G2 == false && G1 == true && D1 == false && D2 == false)
```

```
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral -
    angle_low);
        }

        else if (G2 == true && G1 == true && D1 == false && D2 == false)
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral -
    angle_med);
        }

        else if (G2 == true && G1 == false && D1 == false && D2 == false)
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral -
    angle_high);
        }

        else if (G2 == false && G1 == false && D1 == true && D2 == false)
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral +
    angle_low);
        }

        else if (G2 == false && G1 == false && D1 == true && D2 == true)
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral +
    angle_med);
        }

        else if (G2 == false && G1 == false && D1 == false && D2 == true)
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral +
    angle_high);
        }

        else
        {
            cyhal_pwm_set_period(&pwm_servo_control, period, duty_cycle_neutral);
        }
        t_saved = cyhal_timer_read(&led_blink_timer);

       }

       /* Check if timer elapsed (interrupt fired) and toggle the LED */
       if (timer_interrupt_flag)
       {
           /* Clear the flag */
           timer_interrupt_flag = false;

           /* Invert the USER LED state */
           cyhal_gpio_toggle(CYBSP_USER_LED);
       }

       if (IR_pause == true)
     {
        cyhal_system_delay_ms(10);
```

```
428             IR_pause = false;
429           }
430         }
431 }
432
433
434 /*******************************************************************************
435 * Function Name: timer_init
436 ********************************************************************************
437 * Summary:
438 * This function creates and configures a Timer object. The timer ticks
439 * continuously and produces a periodic interrupt on every terminal count
440 * event. The period is defined by the 'period' and 'compare_value' of the
441 * timer configuration structure 'led_blink_timer_cfg'. Without any changes,
442 * this application is designed to produce an interrupt every 1 second.
443 *
444 * Parameters:
445 *  none
446 *
447 *******************************************************************************/
448  void timer_init(void)
449  {
450     cy_rslt_t result;
451
452     const cyhal_timer_cfg_t led_blink_timer_cfg =
453     {
454         .compare_value = 0,                 /* Timer compare value, not used */
455         .period = LED_BLINK_TIMER_PERIOD,   /* Defines the timer period */
456         .direction = CYHAL_TIMER_DIR_UP,    /* Timer counts up */
457         .is_compare = false,                /* Don't use compare mode */
458         .is_continuous = true,              /* Run timer indefinitely */
459         .value = 0                          /* Initial value of counter */
460     };
461
462     /* Initialize the timer object. Does not use input pin ('pin' is NC) and
463      * does not use a pre-configured clock source ('clk' is NULL). */
464     result = cyhal_timer_init(&led_blink_timer, NC, NULL);
465
466     /* timer init failed. Stop program execution */
467     if (result != CY_RSLT_SUCCESS)
468     {
469         CY_ASSERT(0);
470     }
471
472     /* Configure timer period and operation mode such as count direction,
473        duration */
474     cyhal_timer_configure(&led_blink_timer, &led_blink_timer_cfg);
475
476     /* Set the frequency of timer's clock source */
477     cyhal_timer_set_frequency(&led_blink_timer, LED_BLINK_TIMER_CLOCK_HZ);
478
479     /* Assign the ISR to execute on timer interrupt */
480     cyhal_timer_register_callback(&led_blink_timer, isr_timer, NULL);
481
482     /* Set the event on which timer interrupt occurs and enable it */
483     cyhal_timer_enable_event(&led_blink_timer, CYHAL_TIMER_IRQ_TERMINAL_COUNT,
```

```
                                7, true);

    /* Start the timer with the configured settings */
    cyhal_timer_start(&led_blink_timer);
 }


/*******************************************************************************
* Function Name: isr_timer
********************************************************************************
* Summary:
* This is the interrupt handler function for the timer interrupt.
*
* Parameters:
*    callback_arg    Arguments passed to the interrupt callback
*    event            Timer/counter interrupt triggers
*
*******************************************************************************/
static void isr_timer(void *callback_arg, cyhal_timer_event_t event)
{
    (void) callback_arg;
    (void) event;

    /* Set the interrupt flag and process it from the main while(1) loop */
    timer_interrupt_flag = true;
    speed = count/(period+1)*10000/3;
    count=0;
}

static void gpio_interrupt_handler(void *handler_arg, cyhal_gpio_irq_event_t event)
{
//    gpio_intr_flag = true;
   if (IR_pause == false)
   {
      count ++;
      IR_pause = true;
   }
}

/* [] END OF FILE */
```

7.2 ROS_2 Workspace

GitHub link for ROS2 codebase here, note switch to master branch to see file tree.

7.3 Code for the LiDAR Sensor and the Raspberry Pi

```
%%
%% This is file `.tex',
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
```


```
%% fileerr.dtx  (with options: 'return')
%%
%% This is a generated file.
%%
%% The source is maintained by the LaTeX Project team and bug
%% reports for it can be opened at https://latex-project.org/bugs/
%% (but please observe conditions on bug reports sent to that address!)
%%
%%
%% Copyright (C) 1993-2021
%% The LaTeX Project and any individual authors listed elsewhere
%% in this file.
%%
%% This file was generated from file(s) of the Standard LaTeX 'Tools Bundle'.
%% -----------------------------------------------------------------------
%%
%% It may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either version 1.3c
%% of this license or (at your option) any later version.
%% The latest version of this license is in
%%    https://www.latex-project.org/lppl.txt
%% and version 1.3c or later is part of all distributions of LaTeX
%% version 2005/12/01 or later.
%%
%% This file may only be distributed together with a copy of the LaTeX
%% 'Tools Bundle'. You may however distribute the LaTeX 'Tools Bundle'
%% without such generated files.
%%
%% The list of all files belonging to the LaTeX 'Tools Bundle' is
%% given in the file 'manifest.txt'.
%%
 \message{File ignored}
\endinput
%%
%% End of file '.tex'.
```

References

[1] DSAP0027476 hex inverter Schmitt trigger chip Datasheet