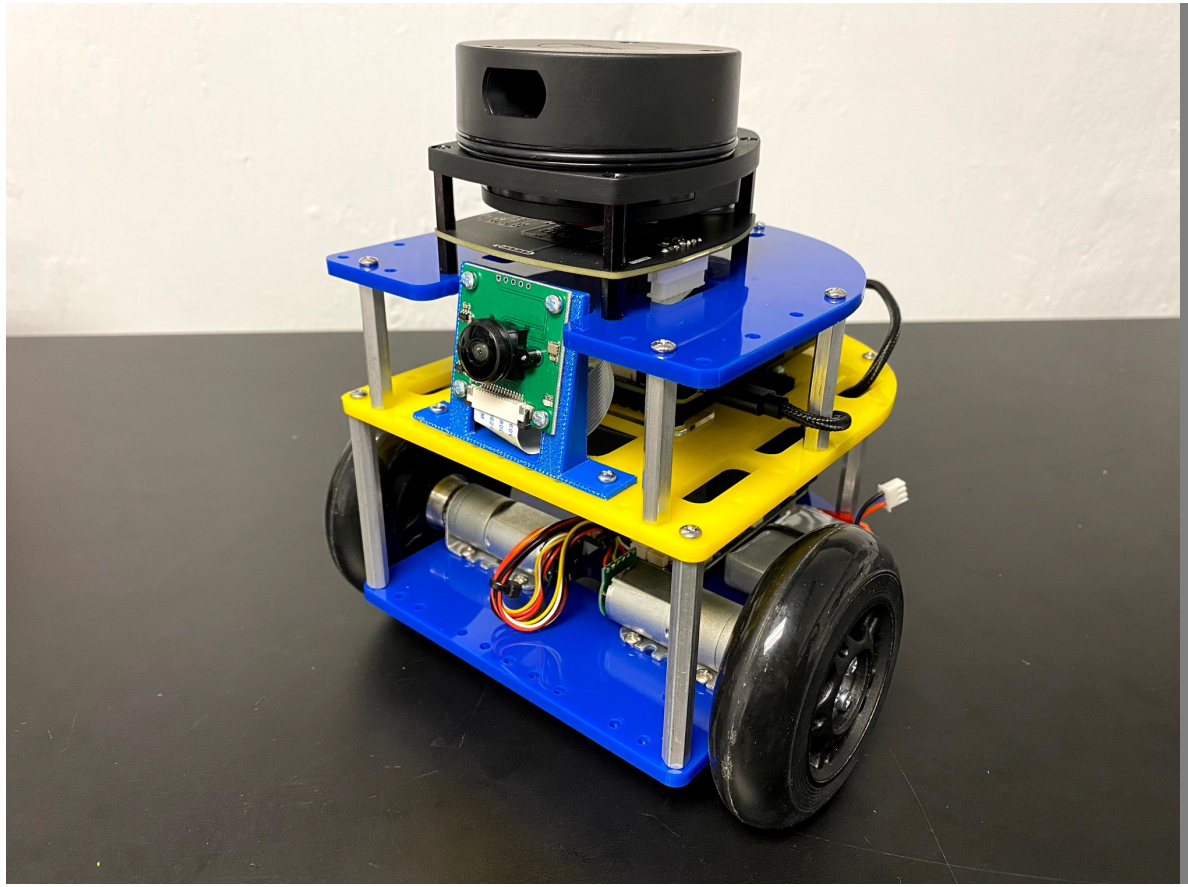
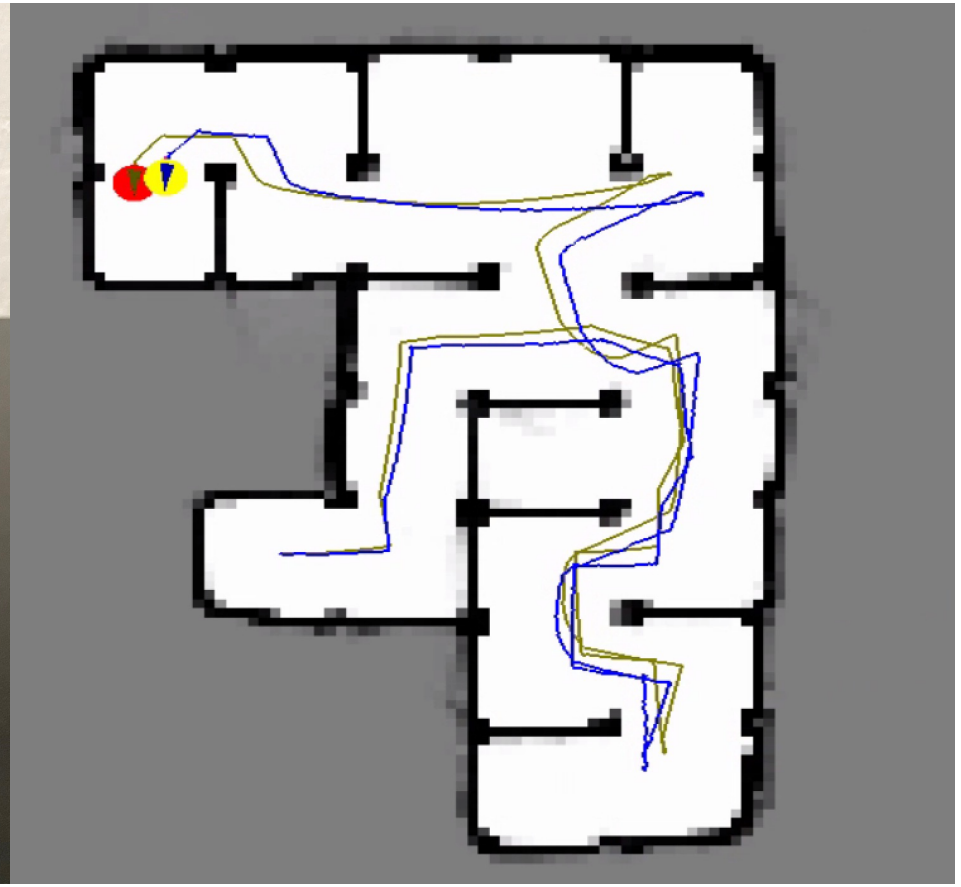
Botlab & MBot Introduction

ROB550

M | ROBOTICS

Botlab





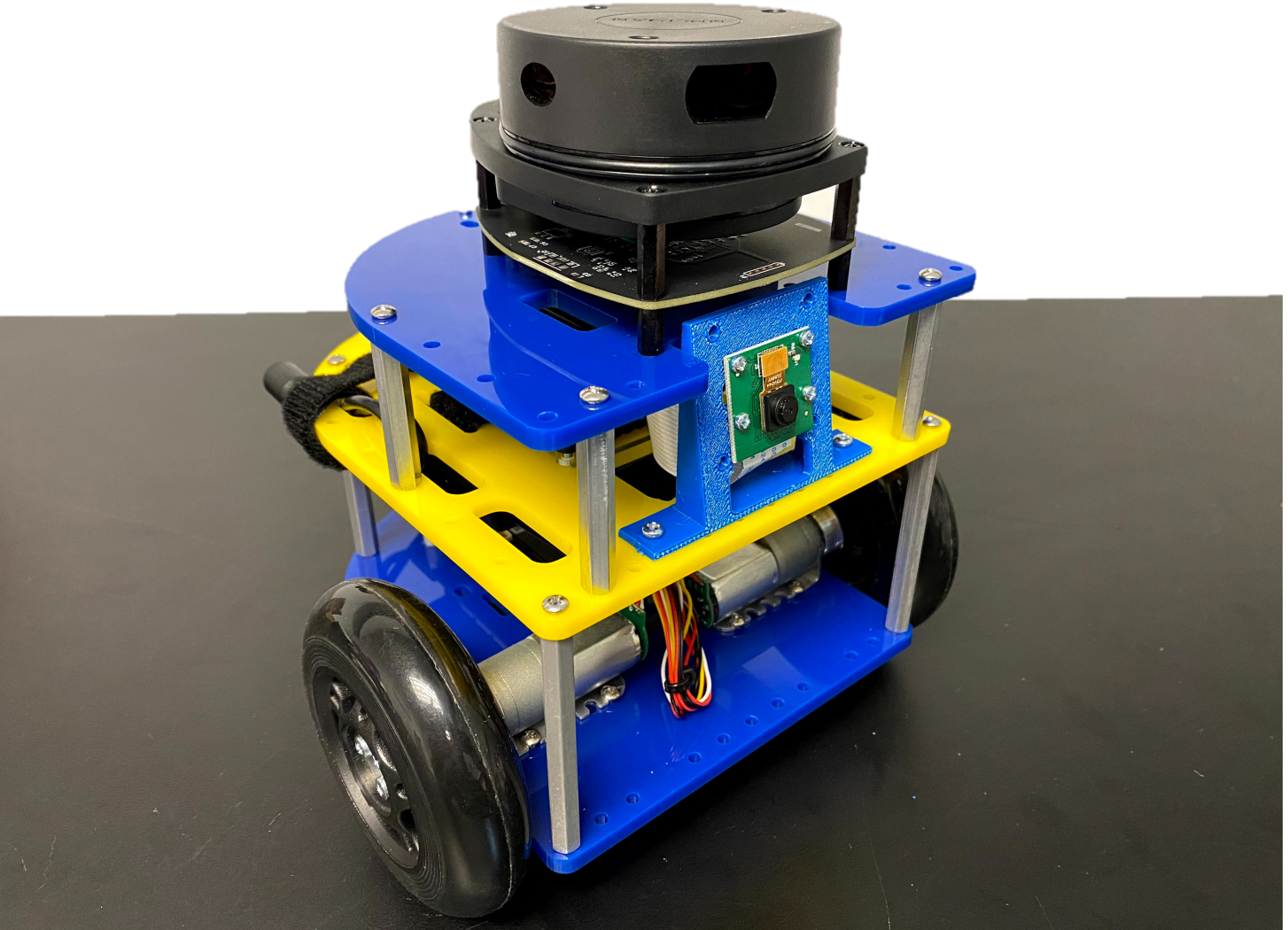
Challenges Part 1

- Characterize behavior of the actuators
- Write a low-level control system on an embedded system
- Develop a system for estimating and controlling relative position using wheel encoders and inertial measurements
- Develop a system for estimating and controlling relative position using a monocular camera and inertial measurements

Challenges Part 2

- Build a 2D mapping system
- Implement a partical filter to estimate location
- Use A* and other search algorithms to navigate
- High level task planning

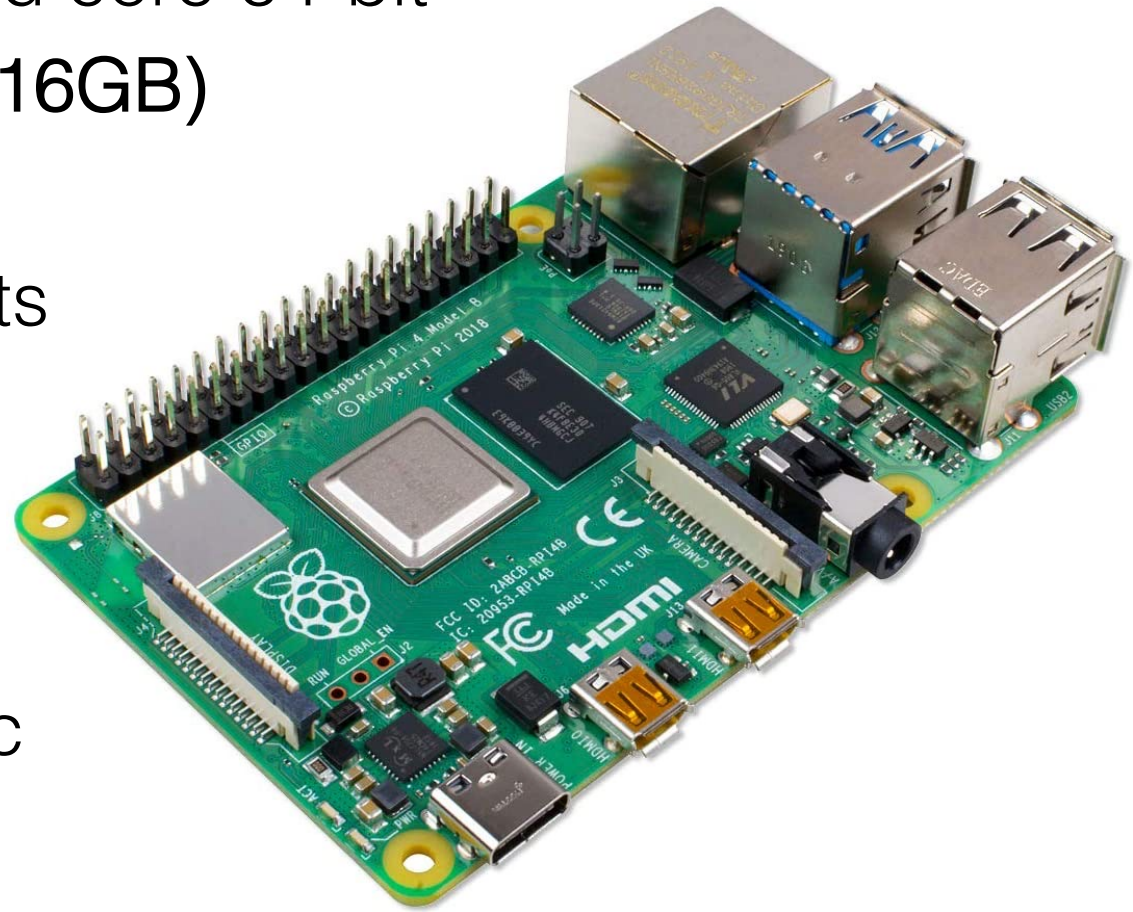
Mbot Mobile Robot Kit



- Bottom
  - Robotics Control Board
  - Brushed DC Motors
  - Magnetic Encoders
  - IMU (Accel, Rate Gyro, Magnetometer)
- Top
  - Raspberry Pi 4B
  - 5MP Camera
  - 2D Scanning LIDAR

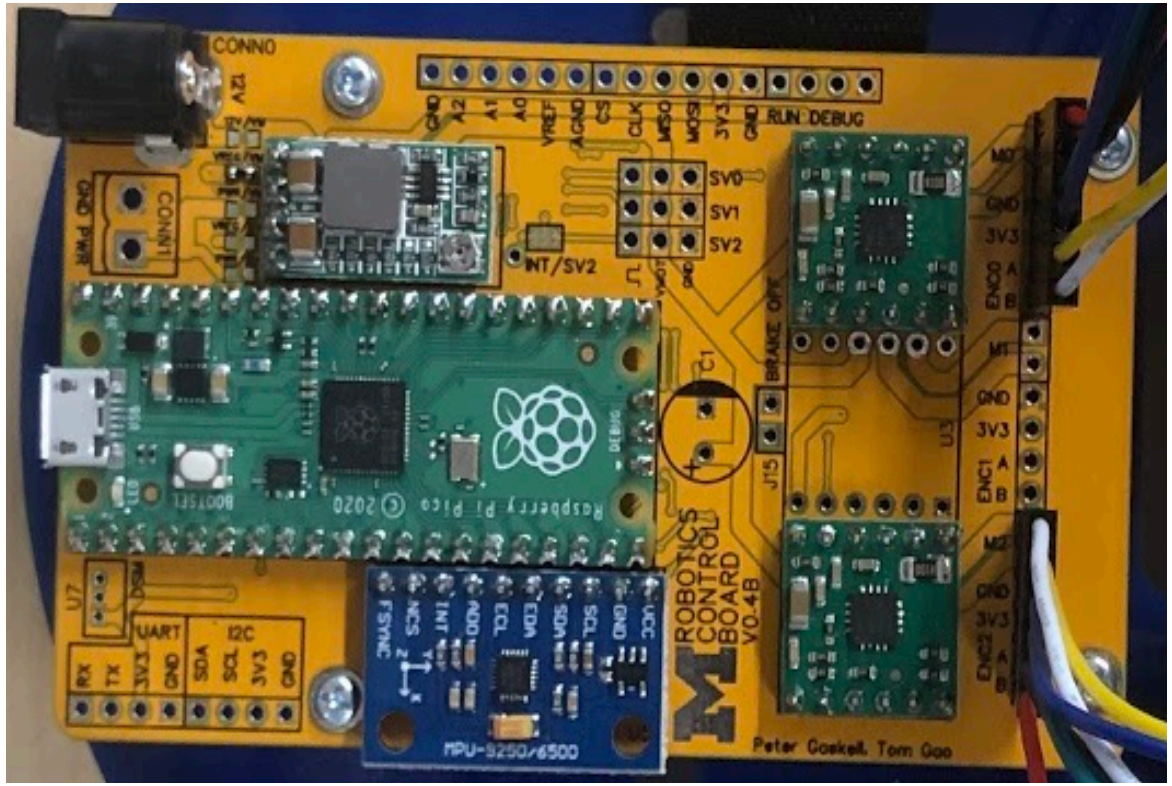
Raspberry Pi 4B

- Broadcom BCM2711 ARM Quad core 64-bit
- Micro-SD card for OS/storage (16GB)
- 2-4GB DDR4-3200 SDRAM
- 2 USB 3.0 ports; 2 USB 2.0 ports
- MIPI CSI camera port
- 2 × micro-HDMI ports
- 40 pin GPIO header
- 802.11 2.4GHz b/g/n + 5Ghz ac
- Gigabit Ethernet



Robotics Control Board

- RPi Pico microcontroller
- 9DOF MPU9250 (low grade)
- DRV8801 brushed motor driver modules
- Encoder interface in PIO

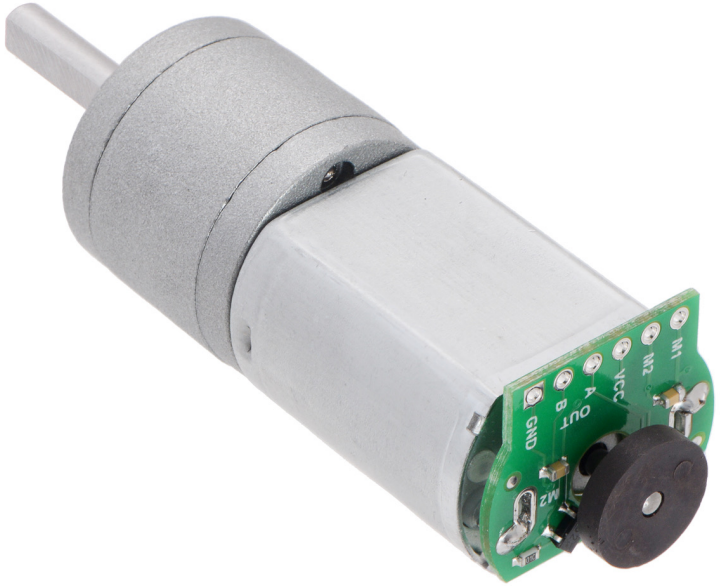


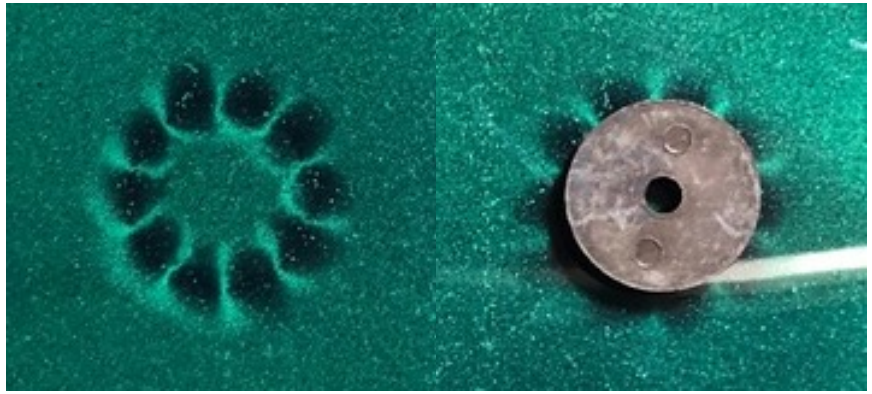
Power System

- 12V 3000mAh Li-Ion Power Bank
  - Powers RPi over 5V USB
  - Powers motors at 12V
  - Charge with wall charger
  - Can charge and use at same time
  - DOES NOT CHARGE VERY FAST
- 2h-4h of operation with batteries fully charged

Motors + Encoders

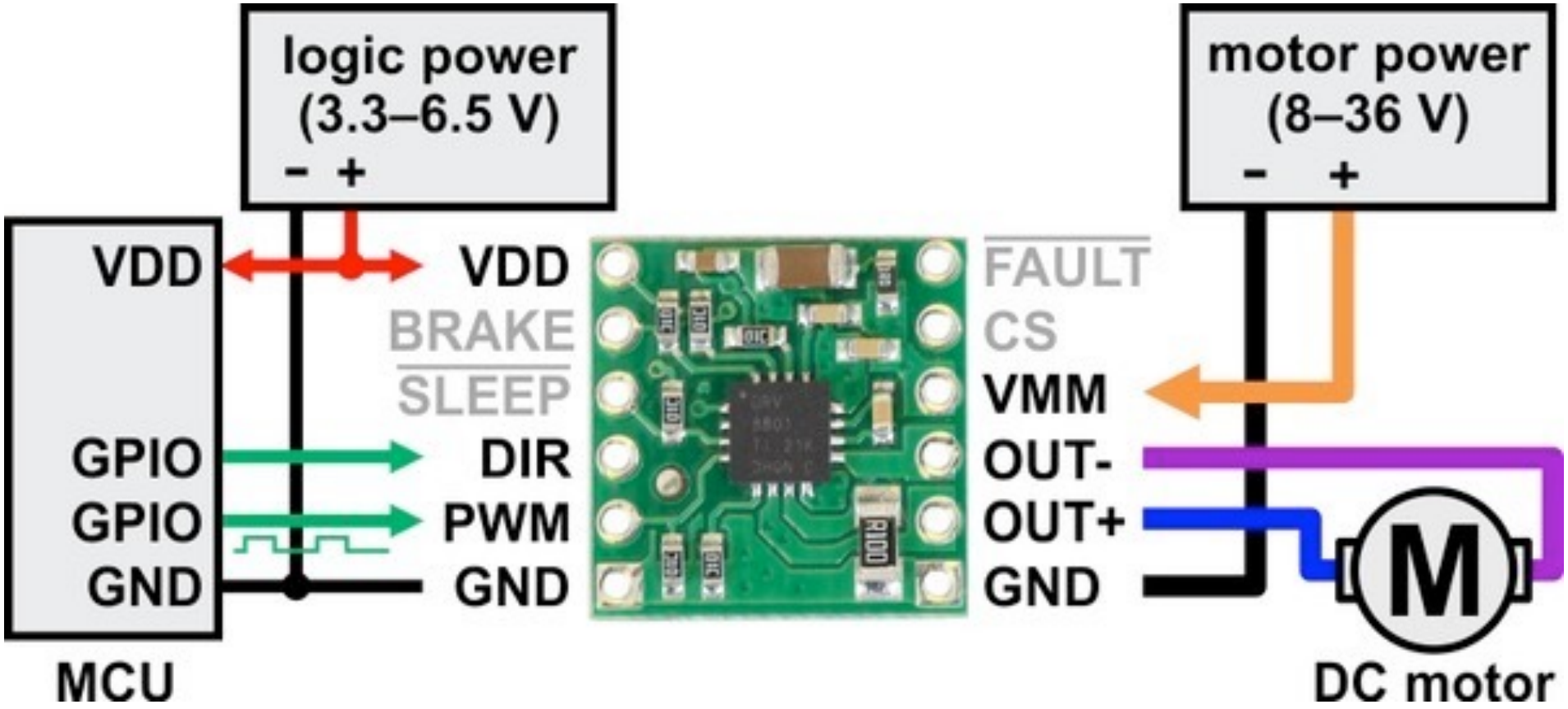
- Pololu 12V Brushed DC gear motors
- 20 count per revolution magnetic encoder
- 78:1 Gear Ratio



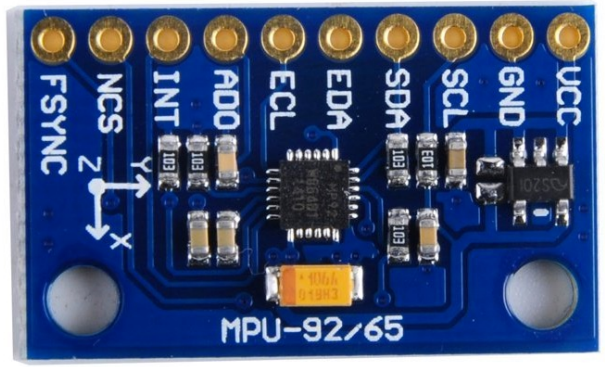


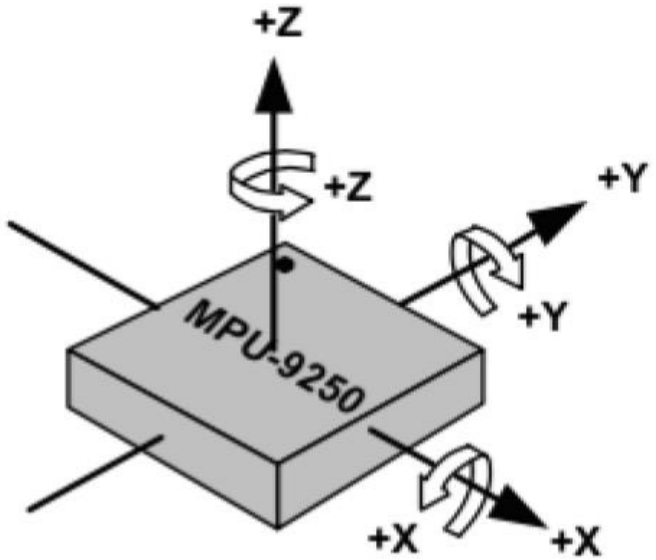
Jiawei Chen

MBot Motor Driver



IMU – MPU9250

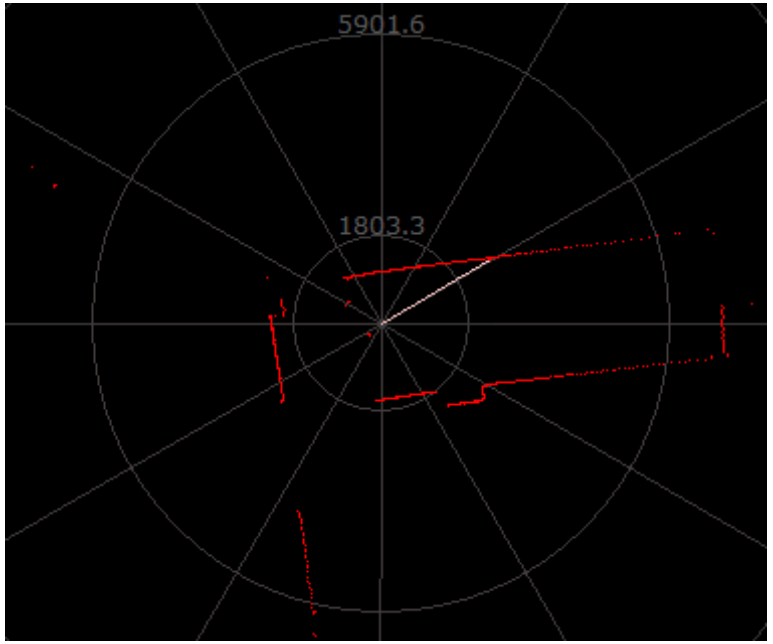


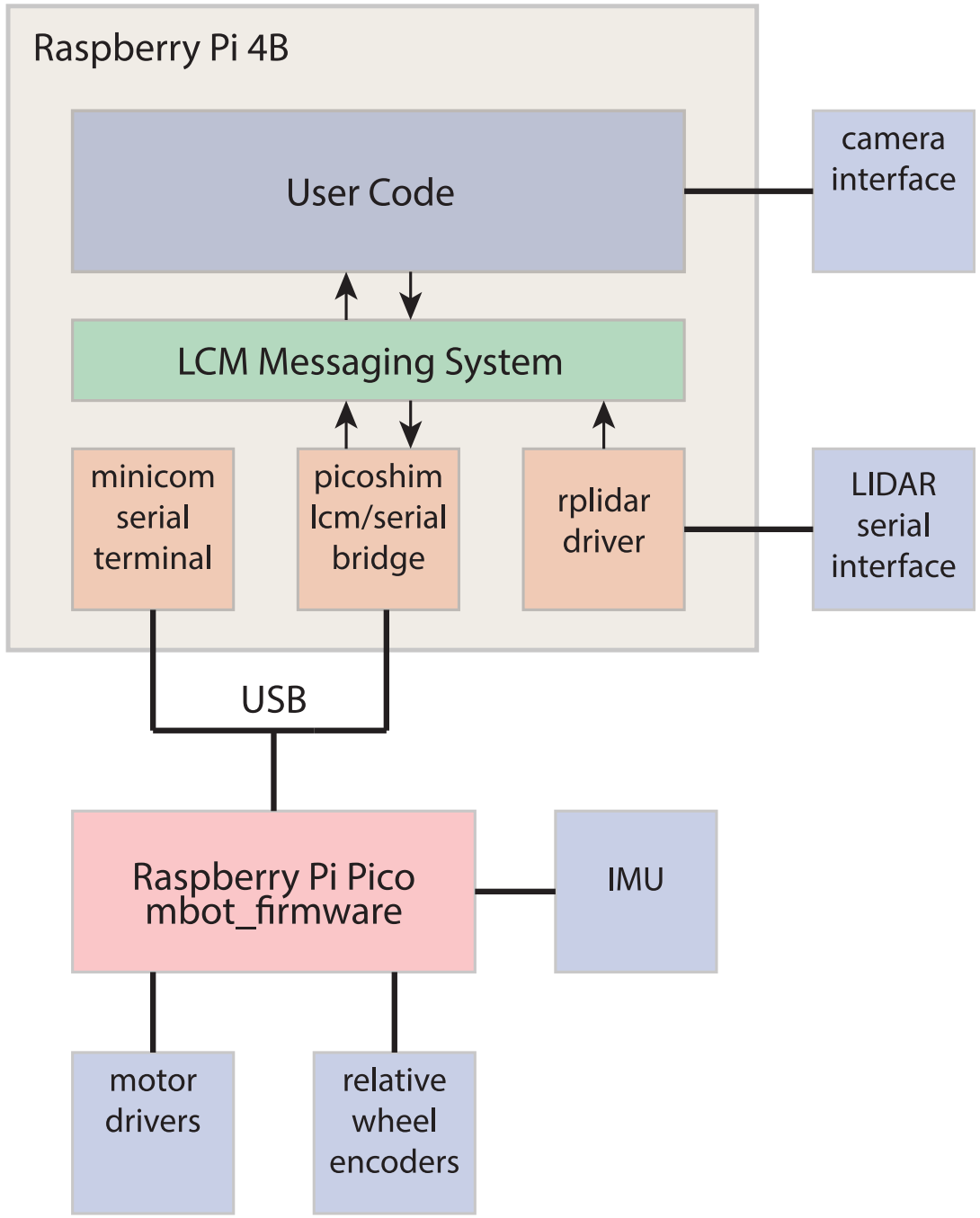


- Accelerometer, Gyroscope, Magnetometer
- Contains data-fusion processor – runs a fusion algorithm (Kalman filter) to determine orientation
- DMP interrupts at fixed rate (200Hz, 100Hz, 50Hz etc.)

LIDAR

- RPLidar A1 2D scanning Lidar
- Parallax sensor (triangulation)
- 0.15m - 12m range
- 0.5mm – 2mm resolution
- 2000 - 8000 Hz sample rate
- 5-10 Hz scanning rate




Raspberry Pi 4B
User Code
camera interface
LCM Messaging System
minicom serial terminal
picoshim lcm/serial bridge
rplidar driver
LIDAR serial interface
USB
Raspberry Pi Pico mbot_firmware
IMU
motor drivers
relative wheel encoders

Lightweight Communications & Marshalling

M | ROBOTICS

Communications and Marshalling

- LCM - https://lcm-proj.github.io/
- Communications: Send data from one program or computer to another program or computer
- Marshalling: Transform memory representation of an object to a form suitable for storage or transmission
- Send predefined messages asynchronously between different programs running on the same processor or across a network
- Messages are Published to a Channel, Subscribers will run a callback function when a message is recieved

Features

- Low-latency inter-process communication
- Efficient broadcast using UDP Multicast
- Type-safe message marshalling
- logging and playback
- No centralized "database" or "hub" -- peers communicate directly, no “ROS master” equivalent
- No daemons (background application) running
- Few dependencies

Platforms & Languages

- C
- C++
- C#
- Go
- Java
- Lua
- MATLAB
- Python

- GNU/Linux
- OS X
- Windows
- Any POSIX-1.2001 system (VxWorks, QNX, BSD)

LCM Type Primitives

Type	Description
`int8_t`	8-bit signed integer
`int16_t`	16-bit signed integer
`int32_t`	32-bit signed integer
`int64_t`	64-bit signed integer
`float`	32-bit floating point
`double`	64-bit floating point
`string`	UTF-8 string
`boolean`	true/false logical value
`byte`	8-bit value

LCM Types

Simple Type: robot_position_t.lcm

```
struct robot_position_t
{
    int64_t utime; // timestamp, in microseconds
    float x;
    float y;
    float theta;
}
```

Variable Array Size: robot_sensor_t.lcm

```
struct robot_sensor_t
{
    int64_t utime; // message timestamp, in microseconds
    int16_t num_samples;
    float sensor_readings[num_samples][2]

}
```

lcm-gen

- Automatically generates source code for a language (command line argument)
- Run with make (check the Makefiles in lcmtypes, python, java)
- You need to include header files for the types you want to use
- Source files contain typedefs/functions for the type (or classes for object-oriented languages)

Publishing

```
robot_pose_t robot_pose;

// set robot_pose to new values
robot_pose.utime = utime_now();
robot_pose.position[0] = x_pos;
robot_pose.position[1] = y_pos;
robot_pose.heading = theta;

// publish robot_pose message
robot_pose_t_publish(lcm, "ROBOT_POSE_CHANNEL", &robot_pose);
```

Subscribe & Handle

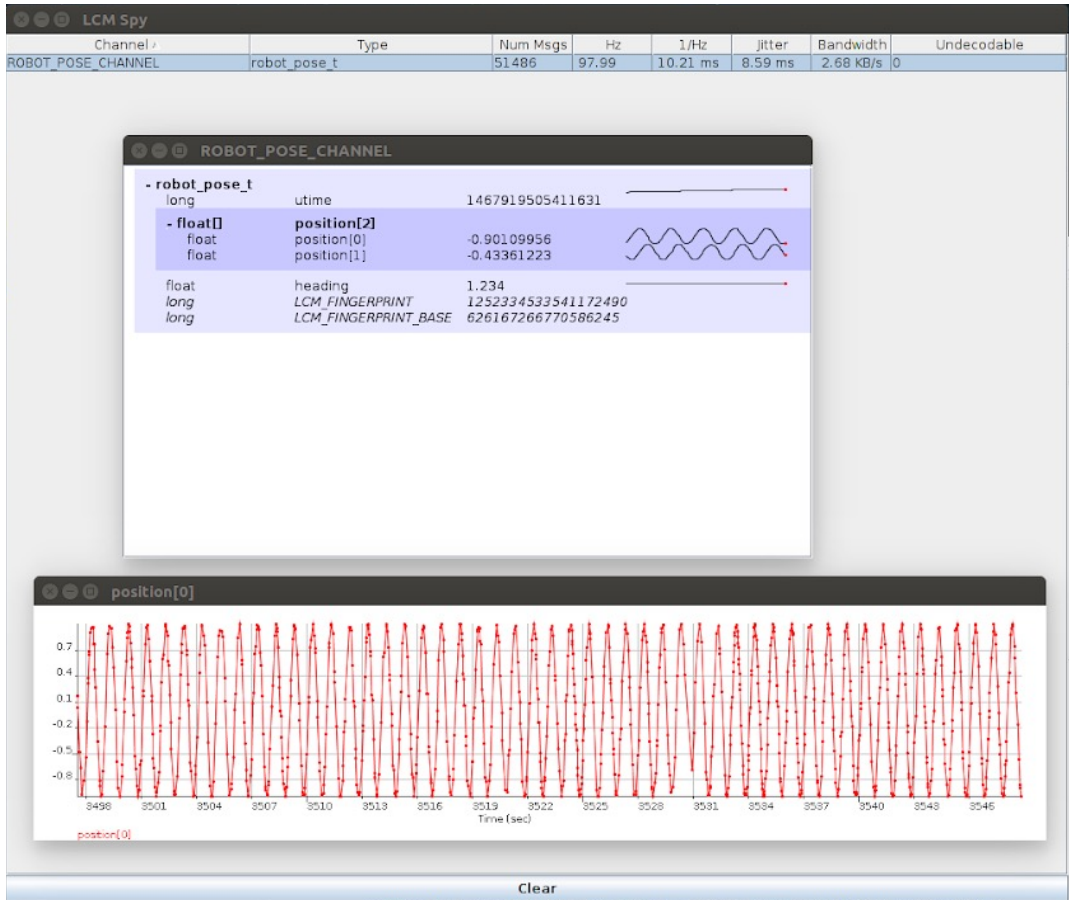
```
#include <lcm/lcm.h>
#include “robot_pose_t.h”
void robot_pose_handler(const lcm_recv_buf_t *rbuf,
                        const char * channel,
                        const robot_pose_t * msg,
                        void * user){
    int64_t timestamp = msg->utime;
    float x_pos = msg->position[0];
    float y_pos = msg->position[1];
    float theta = msg->heading;
    do_something(x_pos, y_pos, theta);
}
int main(){
    lcm_t *lcm = lcm_create("udpm://239.255.76.67:7667?ttl=1");
    if(!lcm){return 1;}
    robot_pose_t_subscribe(lcm, "POSE_CHANNEL", robot_pose_handler, NULL);
    while(1){
        lcm_handle(lcm);
    }
    lcm_destroy(lcm);
    return 0;
}
```

MBot Shim

- Subscribes to LCM messages and transmits them over serial [/dev/ttyACM1]
- Reads from serial and publishes LCM messages
- Messages (for now) need to be defined multiple places

lcm-spy

- Monitor LCM traffic in real time
- Decode messages whose types are in Java .jar file on the classpath
- Use `source setenv.sh` to set classpath
- or add ~/path/to/lcmtypes.jar to .bashrc



lcm-logger & lcm-logplayer

- Logger will log all LCM messages to a log file
- Can be played back with lcm-logplayer
- lcm-logplayer-gui is Java program with extra features
  - Slow-down or speed-up messages
  - Turn on or off channels

Decode a Log File

```
import sys
import lcm
from exlcm import example_t
if len(sys.argv) < 2:
    sys.stderr.write("usage: read-log <logfile>\n")
    sys.exit(1)
log = lcm.EventLog(sys.argv[1], "r")
for event in log:
    if event.channel == "EXAMPLE":
        msg = example_t.decode(event.data)
        print("Message:")
        print("   timestamp   = %s" % str(msg.timestamp))
        print("   position    = %s" % str(msg.position))
        print("   orientation = %s" % str(msg.orientation))
        print("   ranges: %s" % str(msg.ranges))
        print("")
```

Connecting to the MBot

- Need to write pre-made OS image to SD card
- MBot connects to MWireless and has a quasi static IP address
- Setup in lab to connect to MWireless
- Remote desktop protocol to access Ubuntu desktop on MBot
- Development in VSCode
  - Install Remote-SSH extension and run on laptop
  - Develop in VSCode on RPi through remote desktop