## Tasks

1. Create the tool chain and continue to evolve it as Parts are implemented with different technologies.
   1. Github
   2. Python daemons
   3. Arduino source files
   4. C++ make files
   5. Bazel BUILD files
   6. Emulator builds on Mac OS for RPIzero and arduino
2. Create tests for each component that is built with the tool chain.
3. Design and implement communication on the message bus between Arduino and RPIzero.
   1. Implement sensor data stream on Arduino from floor texture sensors
   2. Implement sensor data stream on Arduino from ultrasonic distance and direction sensors
   3. Implement behaviors on Arduino that use the sensor streams
   4. Experiment with sending sensor data over serial connection from Arduino to RPIzero
      * Rpi uses wiringSerial library to read and write data on the serial port
        + C++ program that configures the serial port parameters
        + Sets up the serial object
        + sends data to the serial port
        + receives data from the serial port
        + continuous loop
          - gets data from the serial port and pushes it to MQTT
          - gets data from listeners on MQTT and pushes messages intended for the Arduino through the serial port
      * Arduino uses SoftwareSerial library to read and write data on the serial port.
        + C++ program that configures the serial port parameters
        + Sets up serial object
        + continuous loop
          - gets data from sensor feeds
          - pushes sensor data to serial port
          - receives data from serial port, commands to be executed by the bot
      * Experiment with sending actuator commands over serial connection from RPIzero to Arduino
        + Use power monitor data on Arduino
        + Push data to serial port on Rpi
        + Rpi receives data on serial port
        + Rpi serial io loop pushes data to mqtt with correct channel
        + Rpi listener pulls messages from mqtt and pushes to log
          - nohup mosquitto\_sub -h localhost -t "watts/power" > ~/wattspower.log 2>&1 &
        + Tx/Rx pins on Arduino PD1/PD0
        + Tx/Rx pins on RPIzero GPIO14/GPIP15
   5. Characterize bandwidth requirements of sensor and actuator traffic
4. Design and implement message bus on RPIzero.
   1. Install and test the mosquitto broker  
      sudo apt install mosquitto mosquitto-clients  
      sudo systemctl enable mosquitto /\* enables the broker and restarts after reboot  
      sudo systemctl status mosquitto /\* check status of the broker  
      mosquitto\_sub -h localhost -t "test/message" /\* subscribe to test/message topic  
      mosquitto\_pub -h localhost -t "test/message" -m "Hello, world" /\* test publish a message
   2. use the mosquitto library for client publish and subscribe operations: <https://mosquitto.org/api/files/mosquitto-h.html>
   3. Create a publisher on RPIzero to receive messages from Arduino and place them on the bus
   4. Create a listener on RPIzero to collect published messages as a demonstration of a listener pattern
5. Implement toy TinyML programs to demonstrate RPIzero execution of models.
   1. Characterize resources consumed by models, profiling observables. Optimize.
   2. ML sine
   3. ML watch word
   4. ML face recognition
   5. ML image classification
6. Design and implement observable logging to rpi-in-the-sky.
   1. Implement rpi-in-the-sky API to accept data stream from RPIzero
   2. Implement rpi-in-the-sky web site accessible from browser to present visualizations of the data streams
   3. Implement CPU and microcontroller resource use logging.
   4. Implement observable logging for each Part built.
7. Sensors
   1. FloorTexture Sensor
   2. UltrasonicDistanceDirection Sensor
8. Implement Parts
   1. Implement StayOnRug Part
   2. Implement StayOffRug Part
   3. Implement CrazyIvan Part
   4. Implement NoHeadOnCrash Part
   5. Implement NoSideSwipe Part
   6. Implement CrossToDifferentRug Part
   7. Implement Cruise Part
   8. Implement WaitHere Part
   9. Implement Escape Part
   10. Implement Stuck Part (also known as Strain-Pain Part)
   11. Implement Hunger Part (power monitor)
   12. Implement SeekFood Part
   13. Implement Arbitrator Part (integrates all of the Part behaviors)
9. Prioperceptive Sensors

|  |
| --- |
| Monitor the power consumption of the solar panel, battery and outputs in a solar system  In a typical solar system as shown below (taking the Solar Power Management Module For 12V Lead Acid Battery as an example). Four Gravity: I2C digital Wattmeter are used to monitor the voltage, current and power of the solar panel, battery and two output terminals OUT1 and OUT2, respectively. Each meter is configured by a DIP switch to a different I2C address. They are connected in parallel to the same controller, which records the energy data of each meters (nodes).  For the solar panel, in addition to the voltage, current and power, we maybe more concern about the accumulated electrical energy generated over a period of time. This period may be a day, or it may be several weeks or even months. The time recording can be achieved by adding an RTC module to the controller. Accumulated Power Generation can be obtained by integrating the power with time. For example, sum P power (W) every 1s to obtain the E (electrical) energy (J). This can be converted into another unit Wh by E/3600, or kWh by E / (3.6 × 10 ^ 6) .  For the battery, the wattmeter measures the capacity of the battery by measuring the charge and discharge current. Note that IN+ is directly connected to the positive terminal of the battery, and IN- is connected to the positive input terminal of the BAT IN of the power management module. When the reading is positive, current flows from IN+ to IN-, indicating that the battery is discharging. When the current reading is negative, current flows from IN- to IN+, indicating that the battery is charging. Therefore, the charge and discharge state of the battery can be determined by the positive and negative values of the current. Capacity (Ah), remaining power of the battery is another parameter we may concern. Similar to recording the Accumulated Power Generation, integrating the current with time to get the Capacity (Ah). To obtain a more accurate capacity of the battery, battery can be first fully charged with the power management module and a laptop power adapter (rated output voltage 19V or 20V and power at least 65W). Then, the battery is discharged at one of the OUT terminal until the output is turned off. The Capacity (Ah) is then recorded using the wattmeter at the battery side. Of course, the battery capacity is also related to a number of factors such as the discharge current, the temperature, and the number of cycles the battery has been used. However, the capacity obtained by this method is much more accurate than the nominal capacity of the battery.  For the output terminal OUT, the Accumulated Power Consumption is another power statistic worthy of attention, and its calculation is similar to Accumulated Energy Generation. It is worth noting that Accumulated Energy Generation is usually larger than the sum of the other three Accumulated Power Consumption (assuming the battery is always charging, that is, all three are sourced from solar panels). Excluding the measurement error of the wattmeter, the difference between the two mainly comes from various types of wire and conversion loss, such as: loss of the power management module, loss of the module sampling resistor, and loss of the connection wire and the terminal contact resistor etc. Users can take these losses into account according to the actual application scenarios. |

* 1. Current sensing on main battery
  2. Current sensing on logic batter
  3. Current sensing on Arduino power
  4. Current sensing on RPIzero power
  5. Current sensing on solar cell
  6. Current sensing on motor power x4
  7. Voltage sensing on main battery
  8. Voltage sensing on RPIzero power
  9. Voltage sensing on Arduino power
  10. Voltage sensing on motor power x4

## Experiments

1. Implement a rug-edge detector with the LineTracker sensors  
   Connected the Arduino Uno to the Mac with a long enough cable so Serial window output could be used to monitor sensor outputs. Adjusted the sensitivity potentiometer so that when the three LEDs and sensors are over the rug they return a '1' on their digital inputs and on the hardware floor they return a '0'. A '1' indicates that there is not enough light to trigger the input (we are using the inverse of the digital input). At least one of the three sensors would be 1 as soon as the front of the vehicle was over the rug instead of the hardwood floor. Tested on two rugs.
2. Implement a sensor data stream implementation of StayOnRug
   1. In each loop
      * read digital inputs for FloorTexture, push to message stack with millis() time, mark the readings for the FloorTexture topic to imitate the eventual message bus
        + Use Arduino serial data transfer to listener process on RPIzero that listens on serial line for Prometheus formatted readings and publishes them to Mosquitto message bus mqtt/floortexture channel. Millis() since epoch are the time mark. See Prometheus doc on client libraries for specs of message standard.
        + <https://gist.github.com/andypiper/1218932>
        + <https://5pi.de/2015/02/10/prometheus-on-raspberry-pi/>
        + <https://www.raspberrypi.org/forums/viewtopic.php?t=153385>
      * read distance at direction *i,* push to message stack with millis() time, mark the readings for the DirectionDistanceN, one for each direction *i*, to imitate the eventual message bus
        + cycle through 8 directions of the ultrasonic sensor
        + consider including idle loop() instances so that all 8 direction distances are measured within some targeted repetition time
      * execute StayOnRug() that reads the digital inputs for FloorTexture and pushes primitive moves that create the StayOnRug behavior to the Actuator stack
      * execute Arbitrate() which, at this stage, will simply take the output of StayOnRug primitives on the message bus and pass them along to the Actuator functions. The priority of the behavior outputs is implemented as a table to guide Arbitrator to choose which Actuator values pushed to the message bus to pass on to the Actuators.
3. Implement an Arbitrator as a neural network.
4. Implement an Arbitrator as a neural network frontend to every Part.
5. Implement behaviors for Avoiding, Escaping, Wandering, Seeking and Exploring controlling the action of the drive wheels.
6. Implement behavior for Hunger to look for a power source for recharging.
7. Implement a Strain-Pain-Part to signal that too much current is being drawn.
8. Implement a Stuck-in-the-Mud Part to signal that Alan can't move.

If the Arbitrator is implemented as a table of priorities for behaviors, and if fundamental behaviors with parameters can be combined into behavior sequences, then when a behavior is not reaching its goal, or Watts detects that he’s stuck, he can look for alternative behaviors, and eventually add the alternatives into the Arbitrator table. The Arbitrator in *Mobile Robots* doesn’t have the capacity for allowing multiple behaviors to be triggered at the same priority, but it’s an interesting Experiment to add that capability.