Swarm User Interface Documentation

David Wu davidwu18495@gmail.com
Kyle Dean mwhdean@gmail.com

Objective

Use LEDs and speakers on the Swarmathon rovers to create a swarm user interface. Lights and audio will help the user debug the code that is on the rovers.

Products

→ Blinkstick Flex:
https://www.blinkstick.com/products/blinkstick-flex

Why Blinkstick?

1. One of the main reasons why we chose Blinkstick over other LEDs is because of their ease of use.
    a. Blinksticks have a maximum power draw of 500 mA whereas other LEDs such as the Neopixel have an output of about 2 amps. (USB ports are not able to output 2 amps).
    b. Plug n' play - built in driver so you do not need an arduino to run LEDs.

→ USB Speakers:
https://www.amazon.com/iLuv-Compact-USB-powered-speakers-laptop-Silver/dp/B006EF689M/ref=sr_1_3?ie=UTF8&qid=1502988270&sr=8-3&keywords=iluv+speakers

Project Requirements

- Ubuntu Trusty (14.04)
- ROS Indigo
- ROS control libraries, 3D models, and scripts for NASA Swarmathon robots
- Blinkstick Module for Python
- SFML (Simple and Fast Multimedia Library) to play audio through the speakers via C++

Getting Started

-Make sure you have a working computer with Ubuntu 14.04(many installation guides can be found online…)

a. Install ROS control libraries, 3D models, and scripts for NASA Swarmathon robots.

i. Follow ReadMe.md for full installation guide: https://github.com/BCLab-UNM/Swarmathon-ROS

b. Install arduino for Linux 64 bits
   i. https://www.arduino.cc/en/Main/Software
   ii. Extract file to desired location.
   iii. Open New Terminal: CTRL + ALT + T
   iv. change directory to inside your arduino folder
   v. run the install script with the command $./install.sh

c. Download and upload Swarmathon arduino code to the rover
   i. You can follow the guide here: https://github.com/BCLab-UNM/Swarmathon-Arduino

d. Install Blinkstick Module for Python
   i. Follow installation for Linux: https://github.com/arvydas/blinkstick-python
   ii. Add udev rule to access Blinkstick without root permissions
      1. Open New Terminal: CTRL + ALT + T
      2. run this command: $sudo blinkstick --add-udev-rule

e. Download OpenAl which is needed for SFML
   i. Open New Terminal: CTRL + ALT + T
   ii. run this command: $sudo apt-get install libopenal-dev

f. Install GCC/G++ 4.9 in order to successfully use SFML
   i. Open New Terminal: CTRL + ALT + T
   ii. Follow below commands in terminal

```
$sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$sudo apt-get update
$sudo apt-get install gcc-4.9 g++-4.9
$sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9 60 --slave /usr/bin/g++ g++ /usr/bin/g++-4.9
$sudo apt-get install gcc-4.8 g++-4.8
$sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 60 --slave /usr/bin/g++ g++ /usr/bin/g++-4.8
```

-use $sudo update-alternatives --config gcc to switch between the version of GCC you use.

-make sure it is set on auto mode (usually *option 0*).

Download and Link Required Libraries

1. *Option 1*: Download repository that is ready to go
   https://github.com/MadWarHammer/UH2017

2. *Option 2*: Manually follow step-by-step guide.
   g. Install SFML 2.4.2 for Linux
      i. https://www.sfml-dev.org/download/sfml/2.4.2/
      ii. create a folder at ~/rover_workspace/src/mobility called "cmake_modules"
      iii. Extract SFML download to cmake_modules folder.
      iv. Open SMFL file and go to /SFML-2.4.2/share/SFML/cmake/Modules/
      v. Copy the file "FindSFML.cmake" and place into the cmake_modules folder.

   f. Link Python and SFML in the "CMakeLists.txt" file.
      i. You can find the file at ~/rover_workspace/src/mobility/
      ii. Copy the code below into your CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)

project(mobility)

find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  roscpp
  sensor_msgs
  std_msgs
  random_numbers
  tf
)

find_package(PythonLibs REQUIRED)

include_directories(${PYTHON_INCLUDE_DIRS})

catkin_package(
  CATKIN_DEPENDS geometry_msgs roscpp sensor_msgs std_msgs random_numbers tf
)

include_directories(
  ${catkin_INCLUDE_DIRS}
)
```

```
set(myproject_VERSION_MAJOR 1)
set(myproject_VERSION_MINOR 0)

configure_file(
  "${PROJECT_SOURCE_DIR}/config.h.in"
  "${PROJECT_BINARY_DIR}/config.h"
  )

include_directories("${PROJECT_BINARY_DIR}")

set(EXECUTABLE_NAME "mobility")

add_executable(
  ${EXECUTABLE_NAME}
  src/PickUpController.cpp
  src/DropOffController.cpp
  src/SearchController.cpp
  src/mobility.cpp
)

add_dependencies(${EXECUTABLE_NAME} ${catkin_EXPORTED_TARGETS})

set(CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/cmake_modules" ${CMAKE_MODULE_PATH})

set(SFML_ROOT "/home/swarmie/rover_workspace/src/mobility/cmake_modules/SFML-2.4.2")

find_package(SFML 2.4.2 REQUIRED network audio graphics window system)

if(SFML_FOUND)
  include_directories(${SFML_INCLUDE_DIR})
  target_link_libraries(
  ${EXECUTABLE_NAME}
  ${SFML_LIBRARIES}
  ${SFML_DEPENDENCIES}
  ${catkin_LIBRARIES}
  ${PYTHON_LIBRARIES}
)
else()
  target_link_libraries(
  ${EXECUTABLE_NAME}
  ${catkin_LIBRARIES}
  ${PYTHON_LIBRARIES}
)
endif()

install(TARGETS ${EXECUTABLE_NAME} DESTINATION bin)

include(InstallRequiredSystemLibraries)
set(CPACK_PACKAGE_VERSION_MAJOR "${myproject_VERSION_MAJOR}")
set(CPACK_PACKAGE_VERSION_MINOR "${myproject_VERSION_MINOR}")
include(CPack)
```

iii. make sure the SFML root path matches with your SFML folder location(highlighted in code above).

g. Add code to mobility.cpp to properly play audio and use Blinkstick

i. You can find mobility.cpp at this location: ~/rover_workspace/src/mobility/src/
ii. Include required libraries.

```
#include <Python.h>
#include <SFML/Audio.hpp>
#define RESOURCE_DIRECTORY "/home/swarmie/rover_workspace/resources/sounds/"
```

--The resource directory is going to be where you placed all your sounds files. Make sure the path is correct accordingly.

Adding Audio and Blinkstick Code

1. Pick the sounds you want to use
    a. Gather *.wav* files that you think will help you debug your swarmies
        i. We used notes from the G pentatonic guitar scale for basic states such as *Skid_Steer* and *Rotate*.
        ii. We had distinct sounds whenever the rover would pick up and drop off a cube.
        iii. Since you can use *.wav* files, the possibilities are endless!!!
        iv. Place your gathered files into a new folder(name it anything you want), preferably in your rover_workspace.
        v. If you want to use our *.wav* files, download here: https://github.com/MadWarHammer/UH2017/tree/master/resources/sounds
2. Open mobility.cpp.
    a. Enumerate the sound files for easy reference

```
enum soundNames{Transform, Rotate, SkidSteer, GoHome, ReadyClaw, PickUp, GetBlock, GotBlock, GiveUp, DropOff, Automode, Initialized};
```

    b. Define sound class.

```
class mySoundClass
{
public:
```

```
    mySoundClass()
    {
        soundID = 0;
        size = 0;
        thread = NULL;
    }
    mySoundClass(vector<string*> paths)
    {
        soundID = 0;
        for (unsigned int i=0;i<paths.size();i++)
            addSound(*paths[i]);
    }
    //public functions
    void addSound(string path)
    {
        sf::SoundBuffer *buffer = new sf::SoundBuffer; //a SoundBuffer preloads a sound, aka gets it ready for use. Here
we have an array of them for multiple sound effects.
        sf::Sound *sound = new sf::Sound; //a Sound is given a SoundBuffer so it may be played. Here we have an array of
them for multiple sound effects.
        string temp= RESOURCE_DIRECTORY;
        temp=temp+path; //adds the destination folder path ("../resources/") to the front of the file name.
        if (!buffer->loadFromFile(temp)) //ie: load the 1st SoundBuffer using the 1st soundEffectName into the 1st sound
            cout<<"File failed to load!!!"<<endl;
        sound->setBuffer(*buffer);
        buffers.push_back(buffer);
        sounds.push_back(sound);
        ++size;
    }
    void playSound(unsigned int newSoundID)
    {
        soundID = newSoundID;
       sounds[soundID]->play();
        return;
    }
    void stopSound(unsigned int newSoundID)
    {
        sounds[newSoundID]->stop();
        return;
    }
    unsigned int getSize()
    {
        return size;
    }
    //privates
private:
    //private functions
    void soundThread()
    {
        sounds[soundID]->play();
        return;
    }
    //private member variables
    vector<sf::SoundBuffer*> buffers;
    vector<sf::Sound*> sounds;
    vector<string*> paths;
    sf::Thread *thread;
    unsigned int soundID;
    unsigned int size;
};
```

c. Create your global variables

```
string previousState; //stores previous state-machine state
mySoundClass soundClass;
```

d. In main, add sound files using the sound class.

```
    soundClass.addSound("Transform.wav");
    soundClass.addSound("Rotate.wav");
    soundClass.addSound("SkidSteer.wav");
    soundClass.addSound("GoHome.wav");
    soundClass.addSound("ReadyClaw.wav");
    soundClass.addSound("PickUp.wav");
    soundClass.addSound("GetBlock.wav");
    soundClass.addSound("GotBlock.wav");
    soundClass.addSound("GiveUp.wav");
    soundClass.addSound("DropOff.wav");
    soundClass.addSound("Automode.wav");
    soundClass.addSound("Initialized.wav");
```

--note: we did not implement every sound for our rovers.

e. Play sound and run LED function at each State Machine state.

```
void mobilityStateMachine(const ros::TimerEvent&) {

       ...

        // Select rotation or translation based on required adjustment
        switch(stateMachineState) {

        // If no adjustment needed, select new goal
        case STATE_MACHINE_TRANSFORM: {
            stateMachineMsg.data = "TRANSFORMING";
           if (previousState != "TRANSFORMING")
           {
              TransformLED();
               soundClass.playSound(Transform);
               previousState = "TRANSFORMING";
           }

            // If returning with a target
            if (targetCollected && !avoidingObstacle) {
                soundClass.playSound(GoHome);
                // calculate the euclidean distance between
                // centerLocation and currentLocation
                dropOffController.setCenterDist(hypot(centerLocation.x - currentLocation.x, centerLocation.y -
currentLocation.y));
                dropOffController.setDataLocations(centerLocation, currentLocation, timerTimeElapsed);

                DropOffResult result = dropOffController.getState();
```

```
                if (result.timer) {
                    timerStartTime = time(0);
                    reachedCollectionPoint = true;
                }

                std_msgs::Float32 angle;

                if (result.fingerAngle != -1) {
                    angle.data = result.fingerAngle;
                    fingerAnglePublish.publish(angle);
                   if (previousState != "DROPOFF")
                   {
                       soundClass.playSound(DropOff);
                       previousState = "DROPOFF";
                    }
                }

                if (result.wristAngle != -1) {
                    angle.data = result.wristAngle;
                    wristAnglePublish.publish(angle);
                }

                if (result.reset) {
                    timerStartTime = time(0);
                    targetCollected = false;
                    targetDetected = false;
                    lockTarget = false;
                    sendDriveCommand(0.0,0);

                    // move back to transform step
                    stateMachineState = STATE_MACHINE_TRANSFORM;
                    reachedCollectionPoint = false;
                    centerLocationOdom = currentLocation;

                    dropOffController.reset();
                } else if (result.goalDriving && timerTimeElapsed >= 5 ) {
                    goalLocation = result.centerGoal;
                    stateMachineState = STATE_MACHINE_ROTATE;
                    timerStartTime = time(0);
                }
                // we are in precision/timed driving
                else {
                    goalLocation = currentLocation;
                    sendDriveCommand(result.cmdVel,result.angleError);
                    stateMachineState = STATE_MACHINE_TRANSFORM;

                    break;
                }
            }
            //If angle between current and goal is significant
            //if error in heading is greater than 0.4 radians
            else if (fabs(angles::shortest_angular_distance(currentLocation.theta, goalLocation.theta)) >
rotateOnlyAngleTolerance) {
                stateMachineState = STATE_MACHINE_ROTATE;
            }
            //If goal has not yet been reached drive and maintane heading
            else if (fabs(angles::shortest_angular_distance(currentLocation.theta, atan2(goalLocation.y -
currentLocation.y, goalLocation.x - currentLocation.x))) < M_PI_2) {
                stateMachineState = STATE_MACHINE_SKID_STEER;
```

```
            }
            //Otherwise, drop off target and select new random uniform heading
            //If no targets have been detected, assign a new goal
            else if (!targetDetected && timerTimeElapsed > returnToSearchDelay) {
                goalLocation = searchController.search(currentLocation);
            }

            //Purposefully fall through to next case without breaking
        }

        // Calculate angle between currentLocation.theta and goalLocation.theta
        // Rotate left or right depending on sign of angle
        // Stay in this state until angle is minimized
        case STATE_MACHINE_ROTATE: {
            //soundClass.playSound(Rotate);
            stateMachineMsg.data = "ROTATING";
           if (previousState != "ROTATING")
           {
              RotateLED();
               soundClass.playSound(Rotate);
               previousState = "ROTATING";
           }
            // Calculate the diffrence between current and desired
            // heading in radians.
            float errorYaw = angles::shortest_angular_distance(currentLocation.theta, goalLocation.theta);

            // If angle > 0.4 radians rotate but dont drive forward.
            if (fabs(angles::shortest_angular_distance(currentLocation.theta, goalLocation.theta)) >
rotateOnlyAngleTolerance) {
                // rotate but dont drive  0.05 is to prevent turning in reverse
                sendDriveCommand(0.0, errorYaw);
                break;
            } else {
                // move to differential drive step
                stateMachineState = STATE_MACHINE_SKID_STEER;
                //fall through on purpose.
            }
        }

        // Calculate angle between currentLocation.x/y and goalLocation.x/y
        // Drive forward
        // Stay in this state until angle is at least PI/2
        case STATE_MACHINE_SKID_STEER: {
            //soundClass.playSound(SkidSteer);
            stateMachineMsg.data = "SKID_STEER";
           if (previousState != "SKID_STEER")
           {
              SkidSteerLED();
               soundClass.playSound(SkidSteer);
               previousState = "SKID_STEER";
           }
            // calculate the distance between current and desired heading in radians
            float errorYaw = angles::shortest_angular_distance(currentLocation.theta, goalLocation.theta);

            // goal not yet reached drive while maintaining proper heading.
            if (fabs(angles::shortest_angular_distance(currentLocation.theta, atan2(goalLocation.y - currentLocation.y,
goalLocation.x - currentLocation.x))) < M_PI_2) {
                // drive and turn simultaniously
                sendDriveCommand(searchVelocity, errorYaw/2);
            }
```

```
        // goal is reached but desired heading is still wrong turn only
        else if (fabs(angles::shortest_angular_distance(currentLocation.theta, goalLocation.theta)) > 0.1) {
             // rotate but dont drive
            sendDriveCommand(0.0, errorYaw);
        }
        else {
            // stop
            sendDriveCommand(0.0, 0.0);
            avoidingObstacle = false;

            // move back to transform step
            stateMachineState = STATE_MACHINE_TRANSFORM;
        }


        break;
    }

    case STATE_MACHINE_PICKUP: {
        stateMachineMsg.data = "PICKUP";
        PickUpResult result;
        // we see a block and have not picked one up yet
        if (targetDetected && !targetCollected) {
           soundClass.playSound(PickUp);
           if (previousState != "PICKUP")
           {
               PickUpLED();
                previousState = "PICKUP";
           }

            result = pickUpController.pickUpSelectedTarget(blockBlock);
            sendDriveCommand(result.cmdVel,result.angleError);
            std_msgs::Float32 angle;

            if (result.fingerAngle != -1) {
                angle.data = result.fingerAngle;
                fingerAnglePublish.publish(angle);
            }

            if (result.wristAngle != -1) {
                angle.data = result.wristAngle;

                // raise wrist
                wristAnglePublish.publish(angle);
            }

            if (result.giveUp) {
               soundClass.stopSound(PickUp);
               soundClass.playSound(GiveUp);
                targetDetected = false;
                stateMachineState = STATE_MACHINE_TRANSFORM;
                sendDriveCommand(0,0);
                pickUpController.reset();
            }

            if (result.pickedUp) {
               soundClass.playSound(GotBlock);
                pickUpController.reset();

                // assume target has been picked up by gripper
```

```
                    targetCollected = true;
                    result.pickedUp = false;
                    stateMachineState = STATE_MACHINE_ROTATE;

                    goalLocation.theta = atan2(centerLocationOdom.y - currentLocation.y, centerLocationOdom.x -
currentLocation.x);

                    // set center as goal position
                    goalLocation.x = centerLocationOdom.x;
                    goalLocation.y = centerLocationOdom.y;

                    // lower wrist to avoid ultrasound sensors
                    std_msgs::Float32 angle;
                    angle.data = 0.8;
                    wristAnglePublish.publish(angle);
                    sendDriveCommand(0.0,0);

                    return;
                }
            } else {
                stateMachineState = STATE_MACHINE_TRANSFORM;
            }

            break;
        }

        case STATE_MACHINE_DROPOFF: {
            stateMachineMsg.data = "DROPOFF";
            break;
        }

        default: {
            break;
        }

        } /* end of switch() */
    }
    // mode is NOT auto
    else {
        // publish current state for the operator to see
        stateMachineMsg.data = "WAITING";
    }

    // publish state machine string for user, only if it has changed, though
    if (strcmp(stateMachineMsg.data.c_str(), prev_state_machine) != 0) {
        stateMachinePublish.publish(stateMachineMsg);
        sprintf(prev_state_machine, "%s", stateMachineMsg.data.c_str());
    }
}
```

f. Define functions for LED patterns.

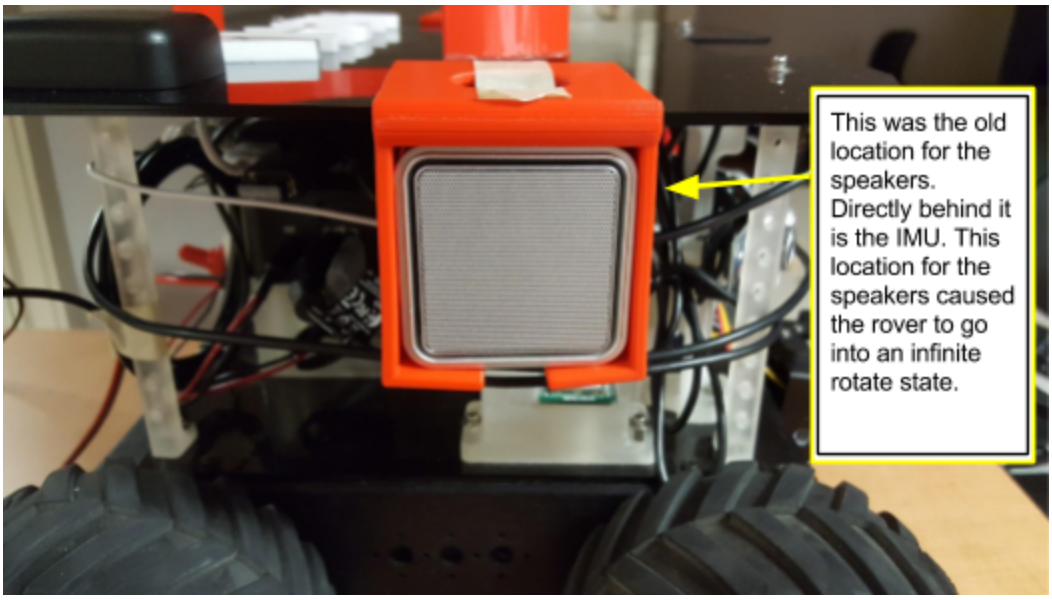
```
void TransformLED() {
    Py_Initialize();
        PyRun_SimpleString("from blinkstick import blinkstick\n"
                           "import time\n"
```

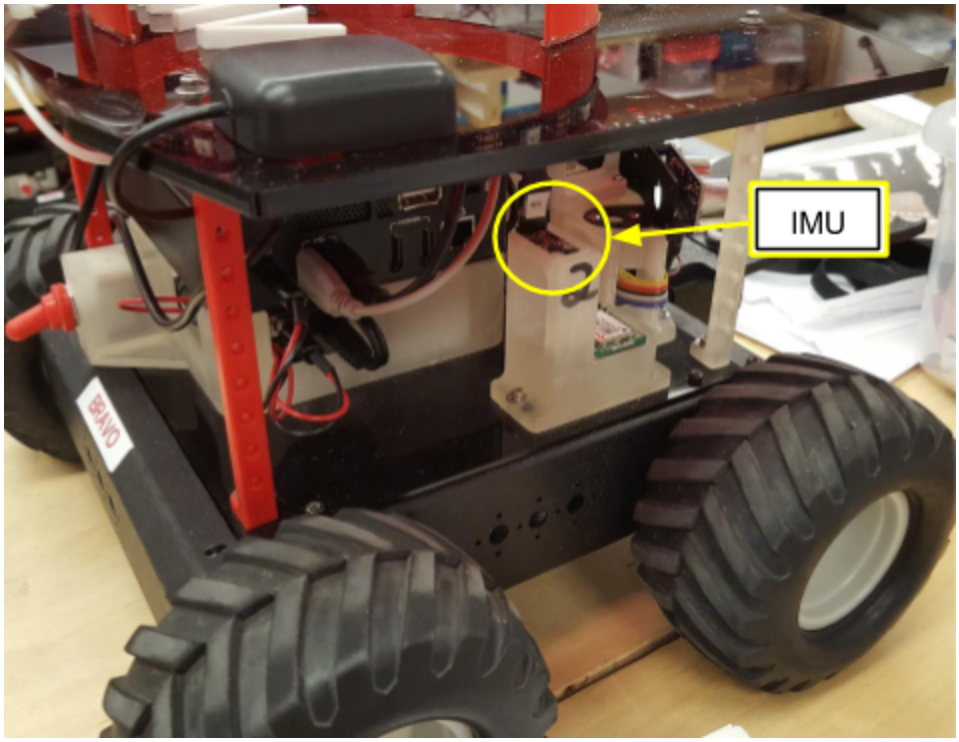
```
                            "bstick=blinkstick.find_first()\n"
                            "bstick.set_led_data(0, [0,255,0] * 16)\n");

    Py_Finalize();
}

void SkidSteerLED() {
    Py_Initialize();
        PyRun_SimpleString("from blinkstick import blinkstick\n"
                            "import time\n"
                            "bstick=blinkstick.find_first()\n"
                            "bstick.set_led_data(0, [0,0,255] * 16)\n");

    Py_Finalize();
}

void RotateLED() {
    Py_Initialize();
        PyRun_SimpleString("from blinkstick import blinkstick\n"
                            "import time\n"
                            "bstick=blinkstick.find_first()\n"
                            "bstick.set_led_data(0, [255,255,0] * 16)\n");

    Py_Finalize();
}


void PickUpLED() {
    Py_Initialize();
        PyRun_SimpleString("from blinkstick import blinkstick\n"
                            "import time\n"
                            "bstick=blinkstick.find_first()\n"
                            "bstick.set_led_data(0, [0,255,255] * 16)\n");
    Py_Finalize();
}

void TurnOffLED() {
    Py_Initialize();
    PyRun_SimpleString("from blinkstick import blinkstick\n"
                       "bstick=blinkstick.find_first()\n"
                       "bstick.set_led_data(0, [0,0,0] * 16)\n");
    Py_Finalize();
}
```

--The *if* statements(highlighted in yellow) are needed for the audio and LED functions so they are not repeatedly called if the previous state was the same as the current state. The only state where an audio function is repeatedly called is the pickup state; this makes it easier to differentiate from the other states.

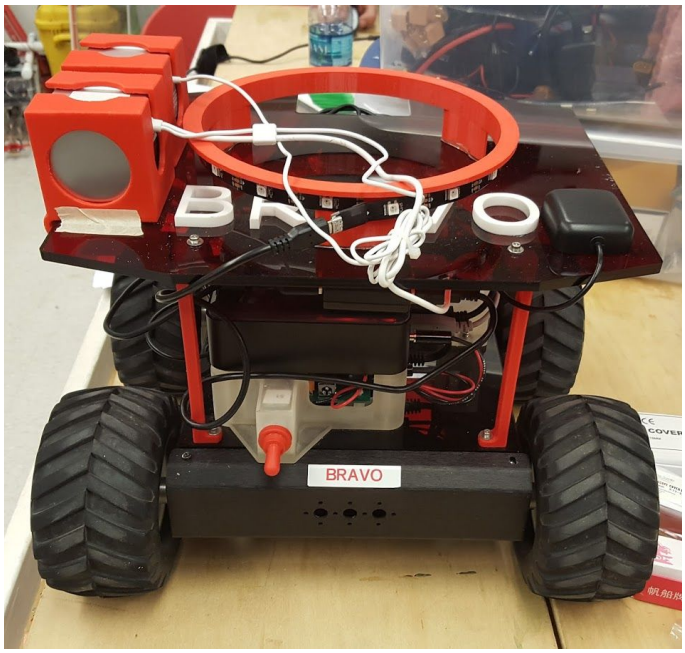
Constraints

2. Placement of speakers
    a. If the speakers are too close to the IMU, the rover goes into an infinite rotate state even when the IMU is calibrated.

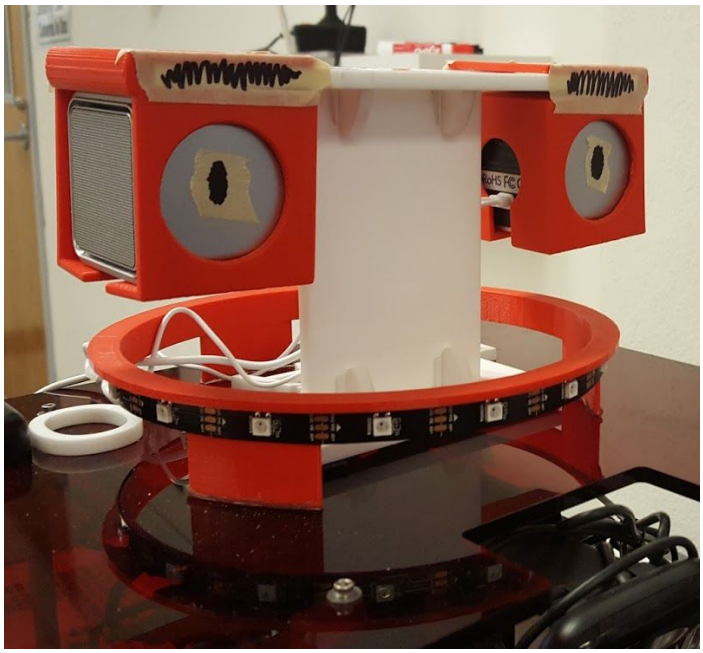




b. We temporarily had to place the speakers on the opposite side of the rover to make sure that our code still worked. We also wanted to see if that was a far enough distance between the speakers and the IMU.



c. A structure was then designed to give the speakers a more permanent home on top of the rover.



3. Blinkstick flex
    a. does not support C++
        i. https://www.blinkstick.com/help/api-implementations
        ii. we had to embed python in mobility.cpp

4. The use of more power.
    a. Adding LEDs and speakers to the rover definitely affects how long the battery lasts on a charge.
    b. There still needs to be testing done to know exactly how much it does affect it.

5. Rover Delay
    a. Since the Blinkstick uses python code there is a small delay when falling through the state machine switch statement.

Additional Links

- Embedding Python: https://docs.python.org/2/extending/embedding.html

- Building SFML Project with CMake: https://github.com/SFML/SFML/wiki/Tutorial%3A-Build-your-SFML-project-with-CMake

- Blinkstick Python Wiki: https://github.com/arvydas/blinkstick-python/wiki

- Swarmathon Website: http://nasaswarmathon.com/

- View the rover in action! https://www.youtube.com/watch?v=9LnGs5e8j-Q

- Project resources: https://docs.google.com/document/d/1Yl4iPly94Tn41ffSP6KDoh3O8bs95pHnEBLhBuvhIFI/edit?usp=sharing

---