Building a Multi-Lateral Cyber-Physical Drone Network using XBee

Yehua Zhang, Dr. Bastian Tenbergen

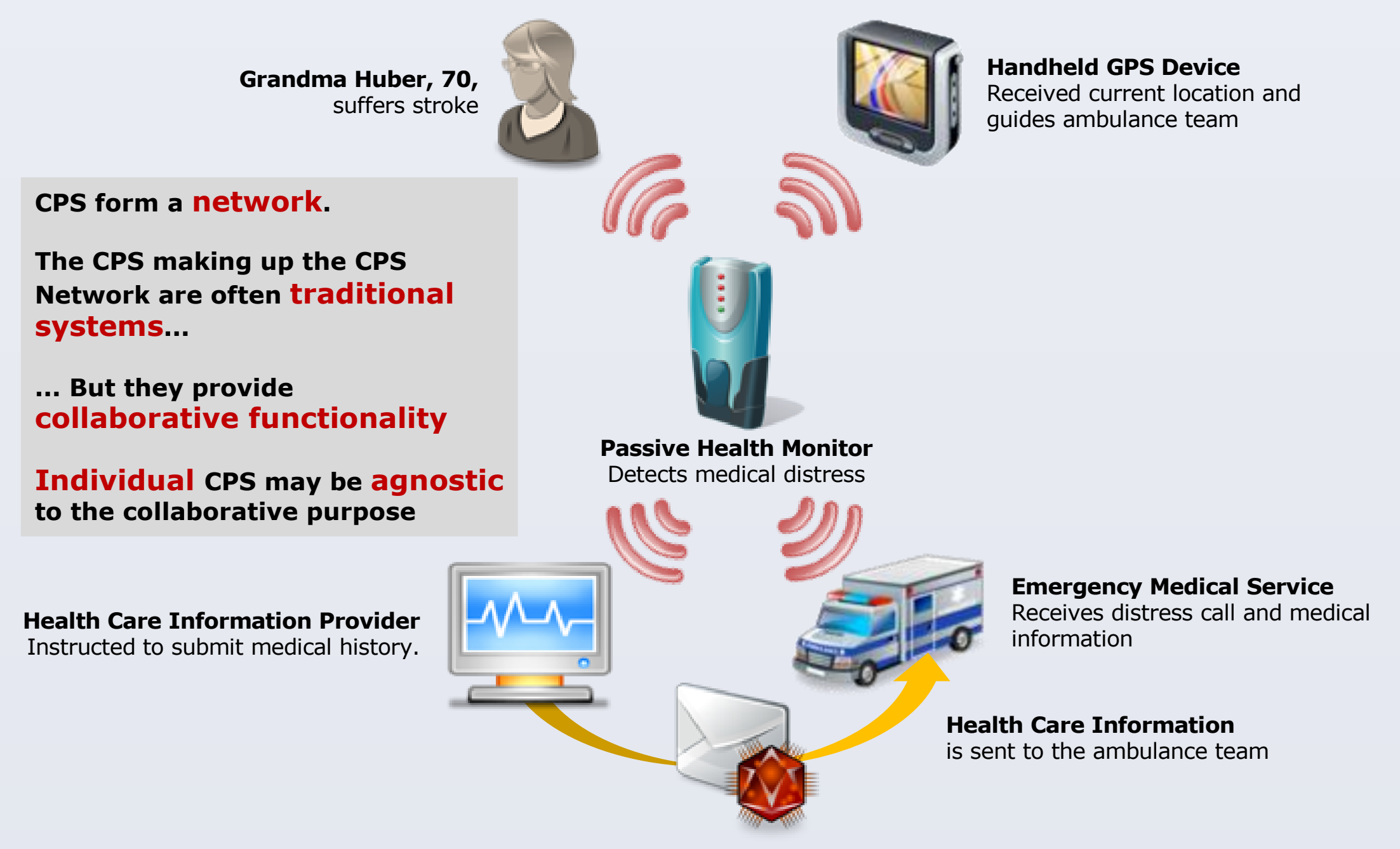
OSWEGO
STATE UNIVERSITY OF NEW YORK

Introduction

In recent years, the software engineering community have used the term “Cyber-Physical System” (CPS) to denote autonomous systems, which collaborate on tasks, no individual CPS can achieve alone ([1-3]). With increasing quality and ubiquity of high-performance communication equipment and computing hardware, previously remote controlled unmanned autonomous vehicles for the military, search and rescue robots, or autonomous cars are becoming more intelligent and can roam the world with minimal human intervention. In consequence, these autonomous CPS are often entrusted with safety-critical behavior. However, this implies new and unforeseen challenges to their development, as the development process must ensure that no human is harmed or injured during operation. In other words, the CPS must be verifiably safe.

Functional Collaboration of CPS

A Smart Health Example similar to [2]


Grandma Huber, 70, suffers stroke
Handheld GPS Device
Received current location and guides ambulance team
CPS form a network.
The CPS making up the CPS Network are often traditional systems…
… But they provide collaborative functionality
Individual CPS may be agnostic to the collaborative purpose
Passive Health Monitor
Detects medical distress
Health Care Information Provider
Instructed to submit medical history.
Emergency Medical Service
Receives distress call and medical information
Health Care Information
is sent to the ambulance team

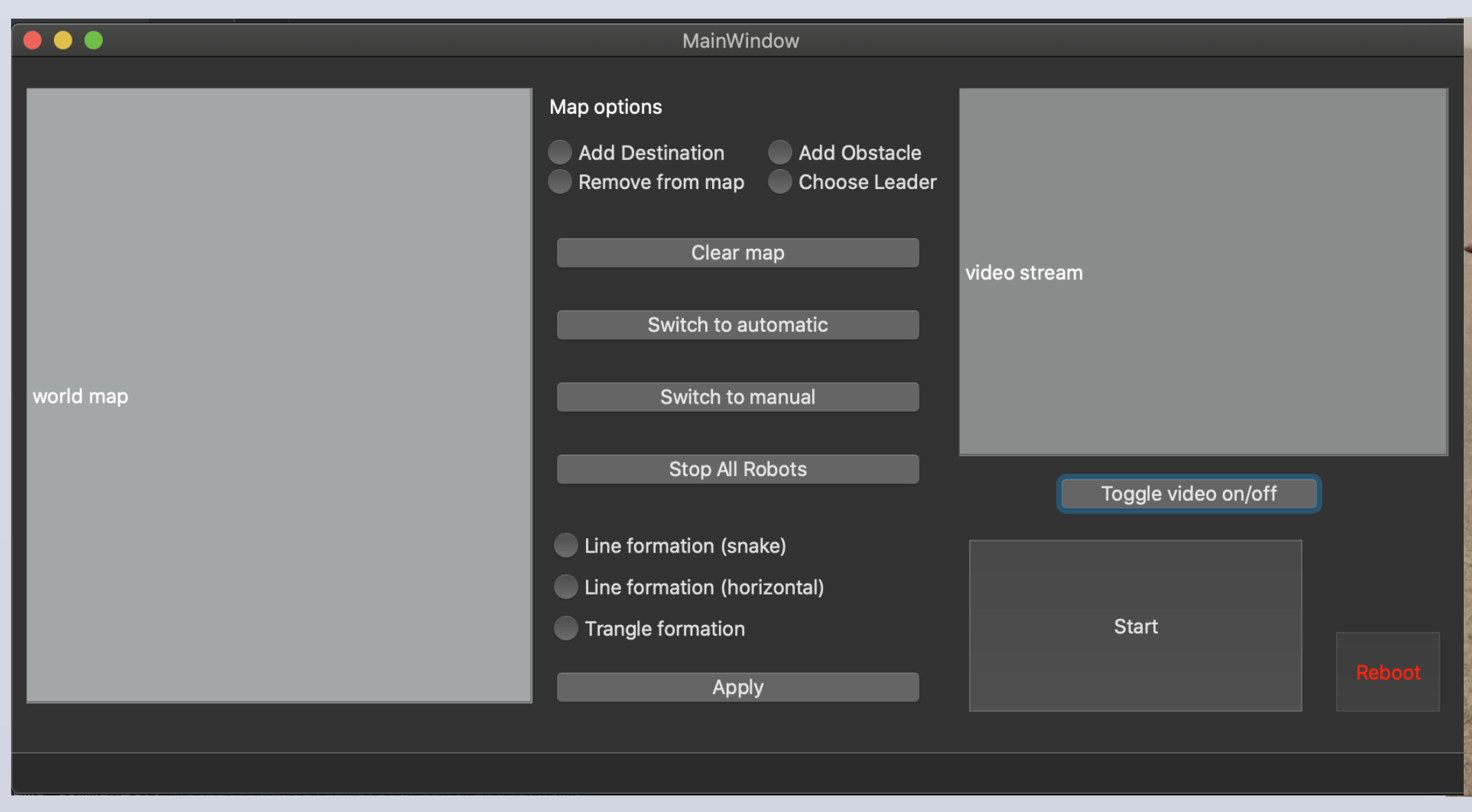
This research program focuses on the challenges during early stages of development of collaborative CPS, when requirements document assertions the future behavior of the CPS. These assertions can then be used to guide safety assessment techniques such as FHA or FMEA. To this end, we used several small instructional robot kits as a simple surrogate platform to investigate the challenges and possible solutions to CPS development.

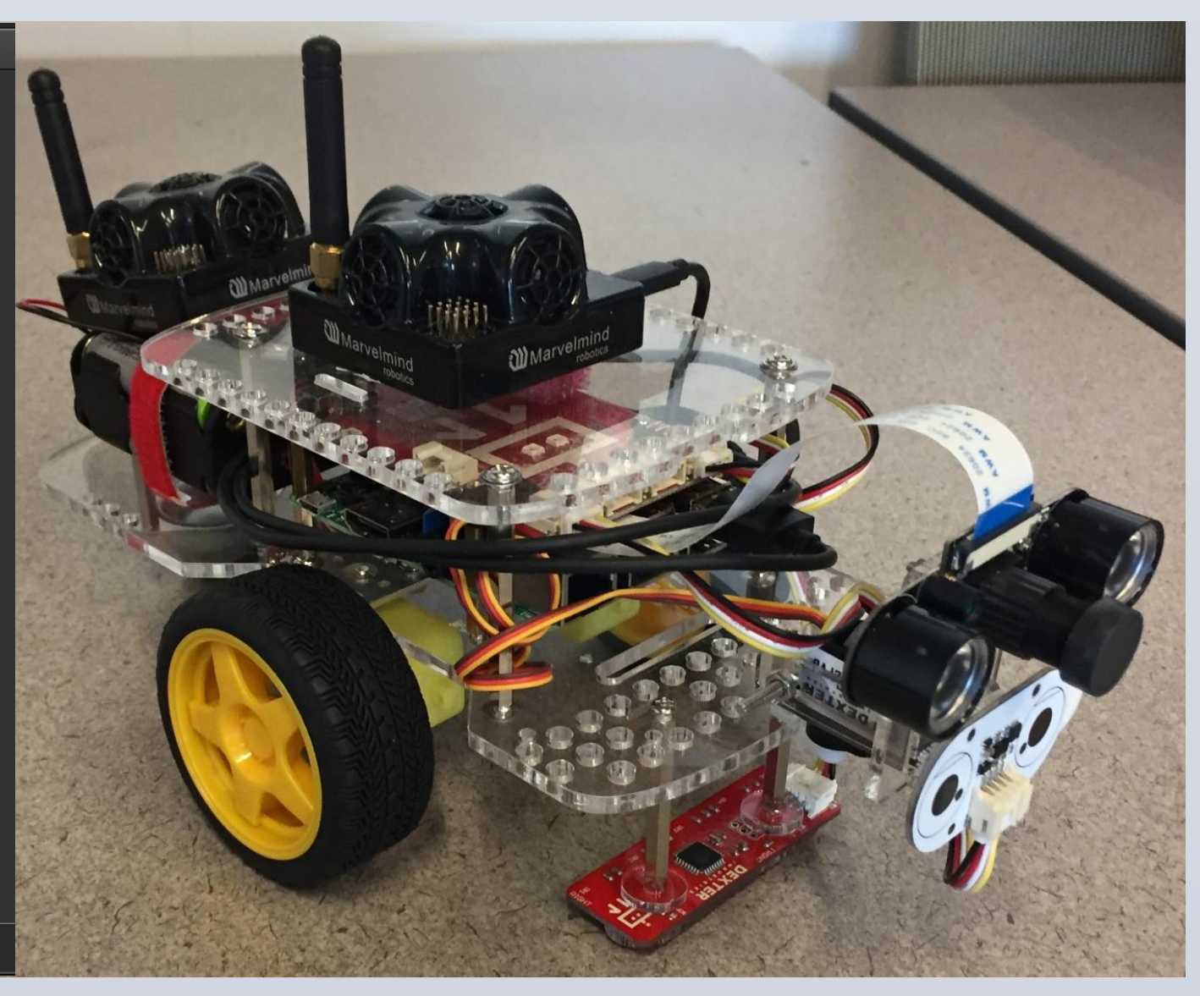
Objectives

To further explore the conceptual challenges of Cyber-Physical Systems development, specifically human-in-the-loop (HiL) control of multiple drones, this project expanded on the previous Student/Faculty Challenge Grant project by incorporating a multi-modal communication system between several robots and a single human operator. To this end, we:

1. Determine the global position of each CPS robot in the environment through ultrasonic trilateralization.
2. Determine each robot’s relative position with regard to other robots.
3. Exchange relative movement information between each robot through Xbee radios.
4. Develop and implement an algorithm to dispatch operator commands within the meshed network.

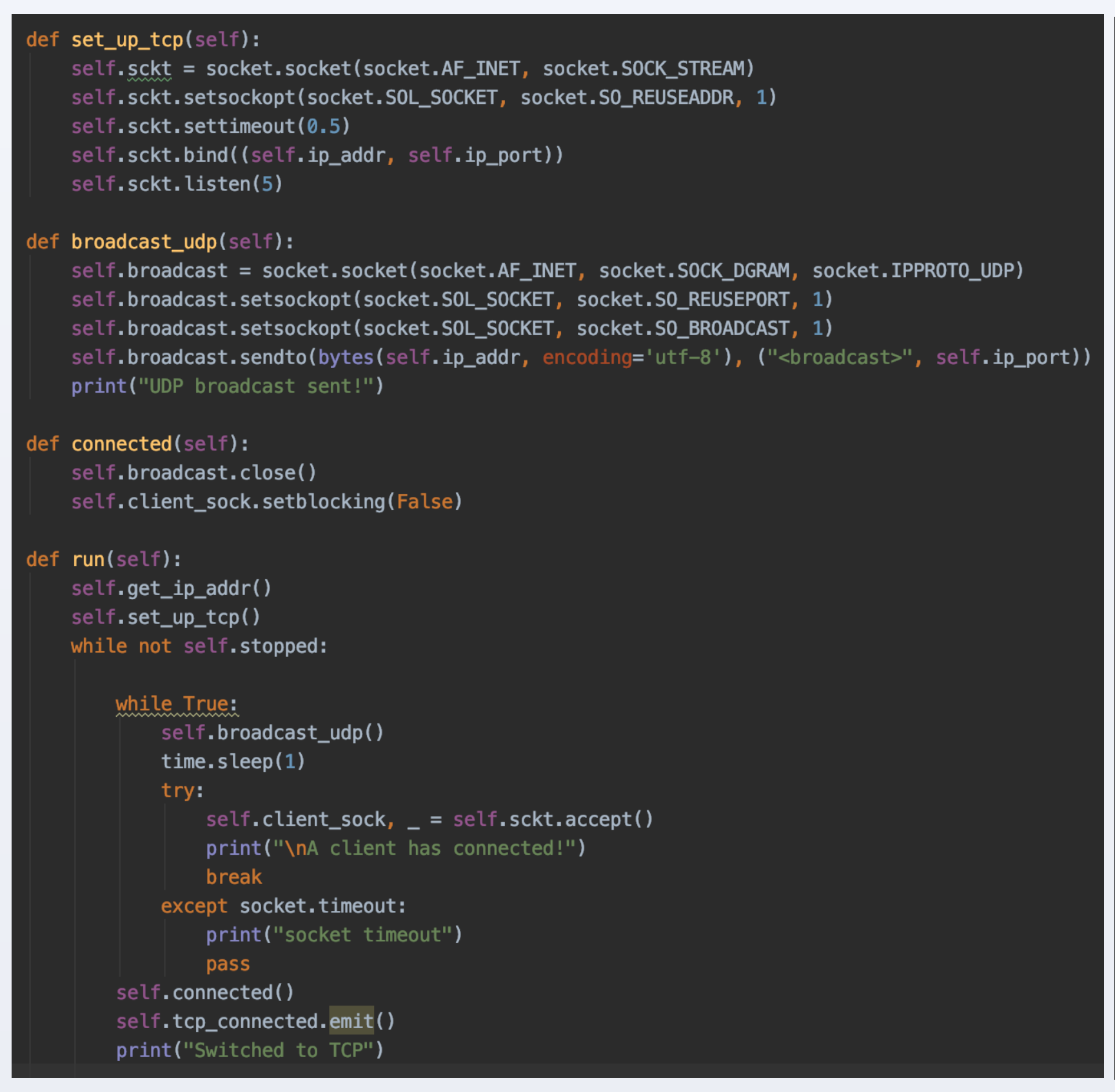
We made use of the ultrasonic indoor position trilaterlization system “RB-Mav-09” by Marvelmind Robotics. And as well as XBee3 radio module which enables the ability that allows us to transmit data through dedicated mesh network.



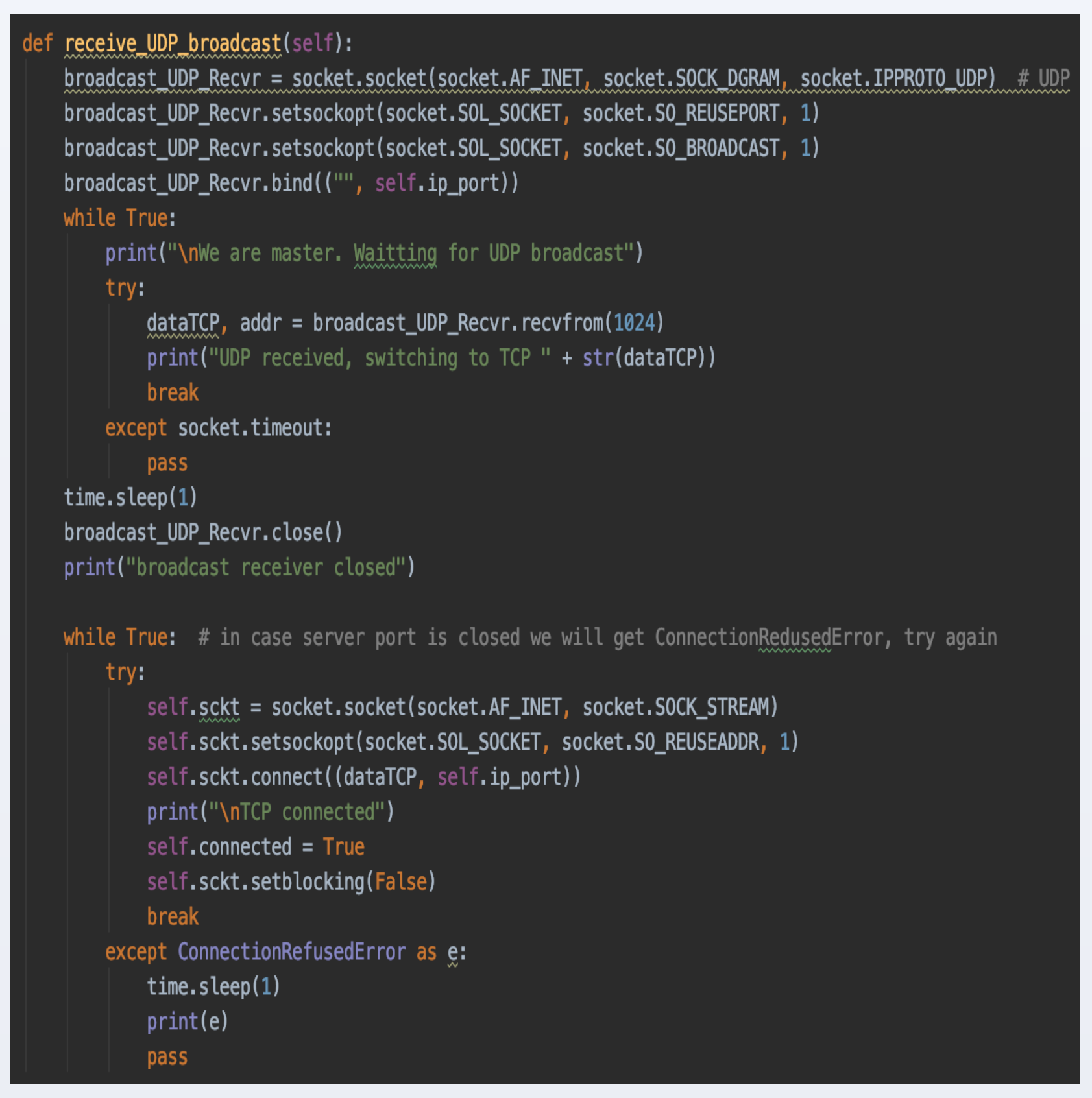


Implementation

It was crucial to maintain a connection between the server and the robot mesh network. To do so, the first thing we did was to reverse the relation between robots and the human operator. In the previous project, we chose the robot as our server to receive commands and data from the human operator. However, this method isn't ideal for our project because in the event of losing connection, the human operator would not know which robot to connect to if there's a change in the remote network.

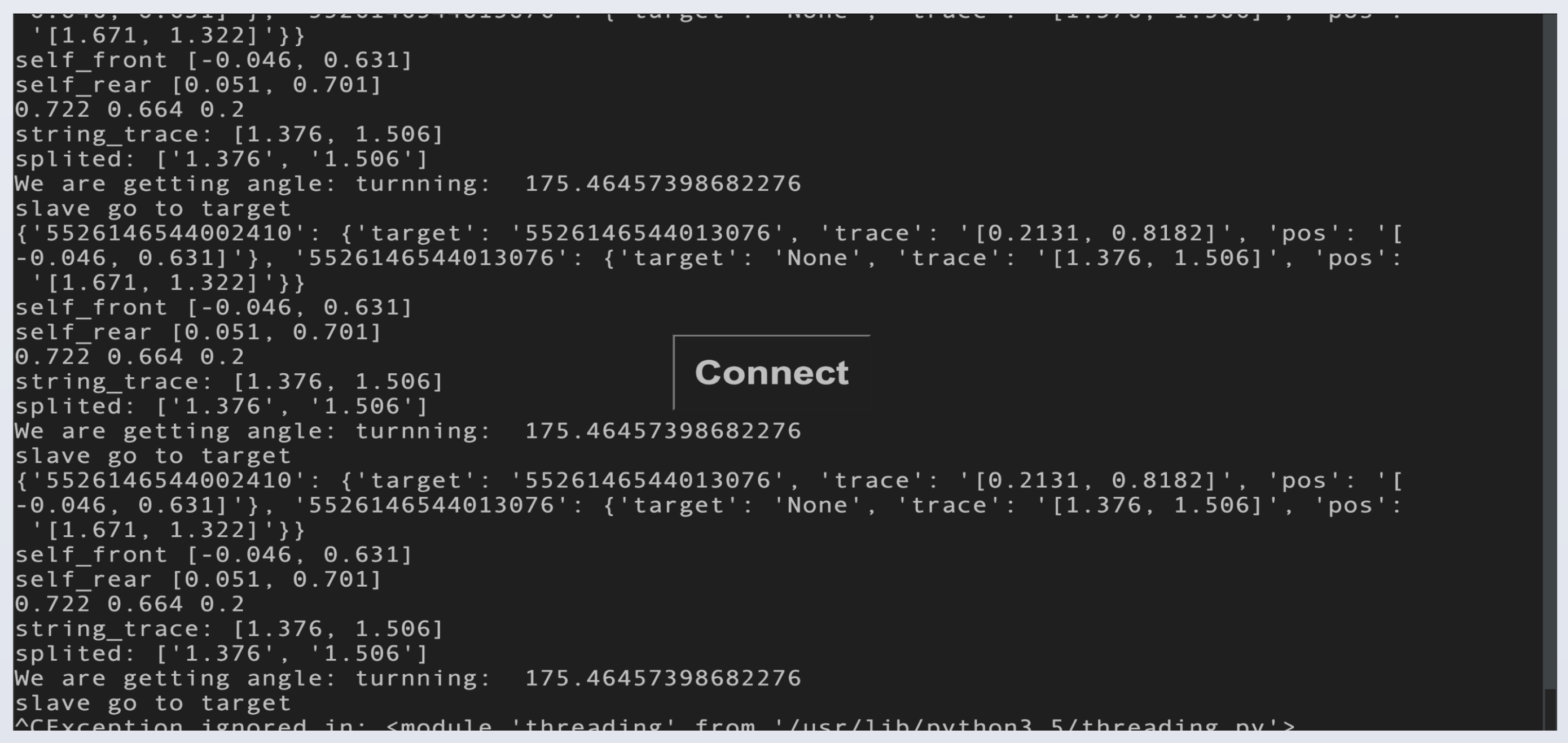


```python
def set_up_tcp(self):
    self.sckt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sckt.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.sckt.settimeout(0.5)
    self.sckt.bind((self.ip_addr, self.ip_port))
    self.sckt.listen(5)

def broadcast_udp(self):
    self.broadcast = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
    self.broadcast.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
    self.broadcast.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    self.broadcast.sendto(bytes(self.ip_addr, encoding='utf-8'), ("<broadcast>", self.ip_port))
    print("UDP broadcast sent!")

def connected(self):
    self.broadcast.close()
    self.client_sock.setblocking(False)

def run(self):
    self.get_ip_addr()
    self.set_up_tcp()
    while not self.stopped:

        while True:
            self.broadcast_udp()
            time.sleep(1)
            try:
                self.client_sock, _ = self.sckt.accept()
                print("\nA client has connected!")
                break
            except socket.timeout:
                print("socket timeout")
                pass
        self.connected()
        self.tcp_connected.emit()
        print("Switched to TCP")
```



```python
def receive_UDP_broadcast(self):
    broadcast_UDP_Recvr = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)  # UDP
    broadcast_UDP_Recvr.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
    broadcast_UDP_Recvr.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    broadcast_UDP_Recvr.bind(("", self.ip_port))
    while True:
        print("\nWe are master. Waitting for UDP broadcast")
        try:
            dataTCP, addr = broadcast_UDP_Recvr.recvfrom(1024)
            print("UDP received, switching to TCP " + str(dataTCP))
            break
        except socket.timeout:
            pass
    time.sleep(1)
    broadcast_UDP_Recvr.close()
    print("broadcast receiver closed")

    while True:  # in case server port is closed we will get ConnectionRedusedError, try again
        try:
            self.sckt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.sckt.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.sckt.connect((dataTCP, self.ip_port))
            print("\nTCP connected")
            self.connected = True
            self.sckt.setblocking(False)
            break
        except ConnectionRefusedError as e:
            time.sleep(1)
            print(e)
            pass
```

A proper network protocol was created as the combination of both UDP and TCP, thus in the event of losing connection to the robot mesh network, our server would be able to start to broadcast UDP signal until an automatically selected master robot is connected from the mesh network, after that the program would switch to TCP for better connection and functionality. These classes serve as the top level of our connection procedure and they always run in a loop so we could recover from the loss of signal safely.



```
 '[1.671, 1.322]'}}
self_front [-0.046, 0.631]
self_rear [0.051, 0.701]
0.722 0.664 0.2
string_trace: [1.376, 1.506]
splited: ['1.376', '1.506']
We are getting angle: turnning:  175.46457398682276
slave go to target
{'5526146544002410': {'target': '5526146544013076', 'trace': '[0.2131, 0.8182]', 'pos': '[
-0.046, 0.631]'}, '5526146544013076': {'target': 'None', 'trace': '[1.376, 1.506]', 'pos':
 '[1.671, 1.322]'}}
self_front [-0.046, 0.631]
self_rear [0.051, 0.701]
0.722 0.664 0.2
string_trace: [1.376, 1.506]
splited: ['1.376', '1.506']
We are getting angle: turnning:  175.46457398682276
slave go to target
{'5526146544002410': {'target': '5526146544013076', 'trace': '[0.2131, 0.8182]', 'pos': '[
-0.046, 0.631]'}, '5526146544013076': {'target': 'None', 'trace': '[1.376, 1.506]', 'pos':
 '[1.671, 1.322]'}}
self_front [-0.046, 0.631]
self_rear [0.051, 0.701]
0.722 0.664 0.2
string_trace: [1.376, 1.506]
splited: ['1.376', '1.506']
We are getting angle: turnning:  175.46457398682276
slave go to target
^CException ignored in: <module 'threading' from '/usr/lib/python3.5/threading.py'>
```

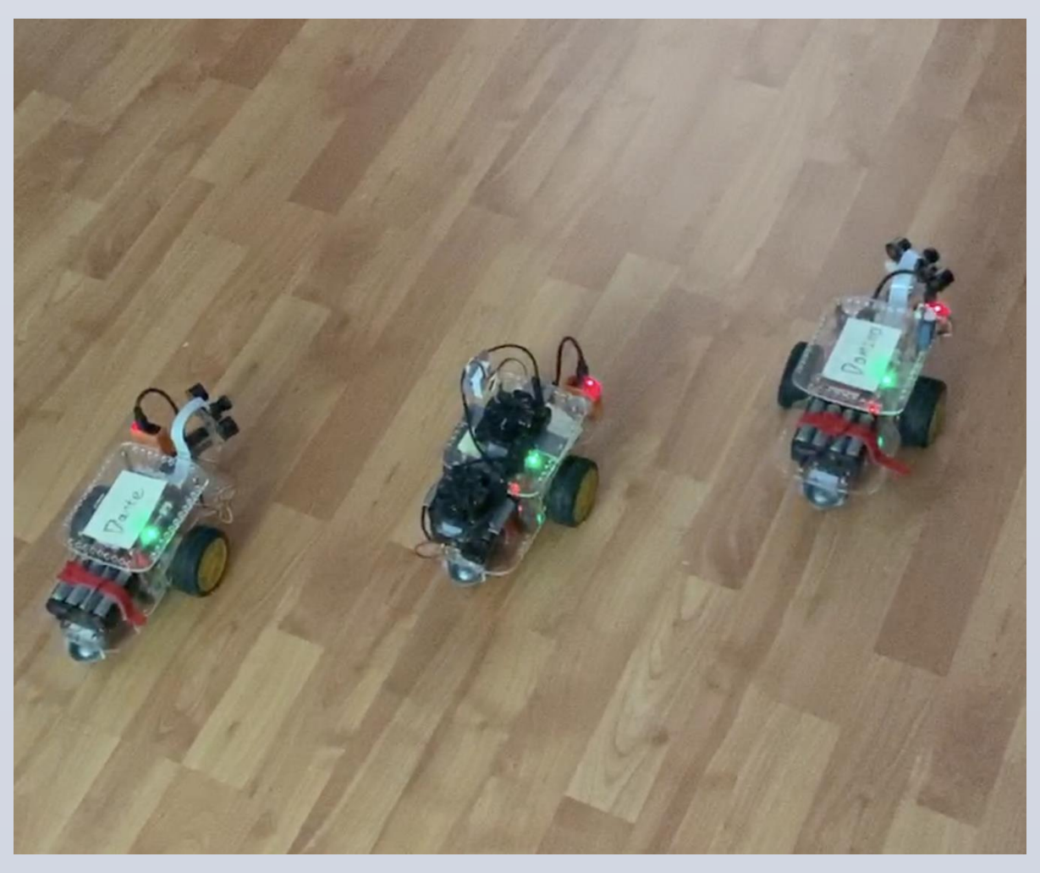
A formation class "formation.py" was created to allows our robots squadron to move in a certain formation and maintain their respective safe distance to each other. We used our XBee mesh network to share and respond each robots’ position data. This is shown in the image below, in which the robots automatically change formation according to their relative position (**click any image for a demo video**).

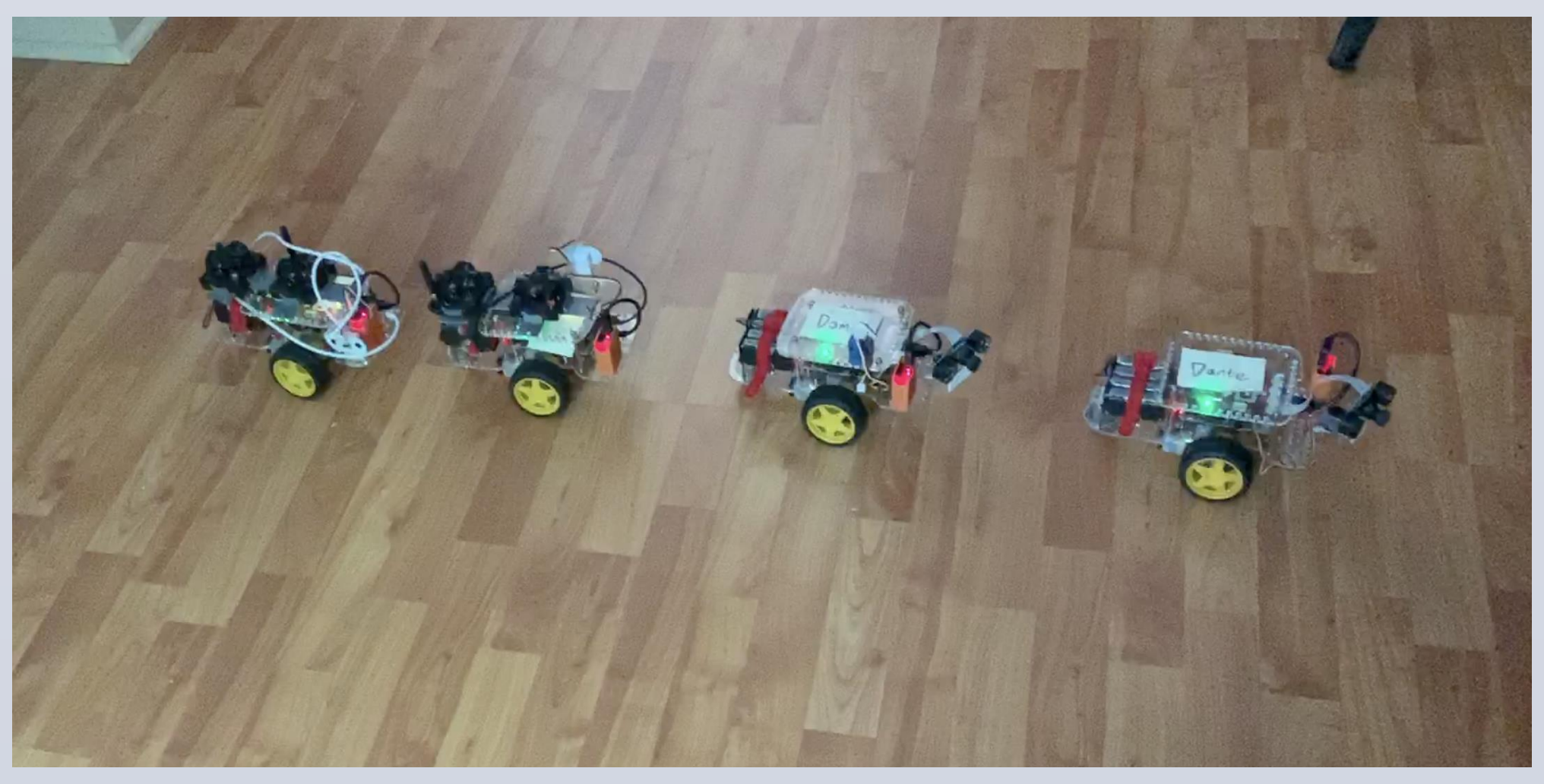
Accomplishments

Using this research, we successfully added more features on previous platform to study the development of autonomous behavior in cyber-physical drones during early stages of development. Moreover, it aids in the identification of challenges, pitfalls, and interaction modes in human-in-the-loop autonomous vehicle control, i.e. when a human must be informed about the possible safety-critical decisions made by a robot during autonomous control mode.

To this end, we implemented the following features:

- Customized network protocol allowing to recover from loss of signal;
- Universal GUI template to improve future updates;
- A mesh network layer to abstract from single-robot communication and enable multi-robot coordination;
- Formation capability to enable robots to move as seemingly one unit, depending on commands received from a single user.





Contact

Web https://github.com/tenbergen/CPS-Rover/tree/master/dante/Formation

Email yzhang9@oswego.edu | bastian.tenbergen@oswego.edu

References

1. Wolf, W. (2009) Cyber-physical Systems. IEEE Comp. 42(3).
2. Broy, M.; Cengarle, M.; Geisberger, E. (2012) Cyber-Physical Systems: Imminent Challenges. Monterey Workshop.
3. Daun, M.; Salmon, A.; Tenbergen, B.; Weyer, T. (2015) Today’s Challenges and Potential Solutions for the Engineering of Collaborative Embedded Systems. 2nd Intl. EITEC WS.