Maritime RobotX Challenge 2022: Heartbeat Message

Sitha Sinoun

Introduction

The Maritime Robotx challenge 2022 requires a communications protocol to report vehicle status along with the participation and completion of chosen autonomous challenges/tasks. The notification system is built to send packets in the format of a NVEA-like sentence message back to the Technical Director's Base/IP. The primary message is the heartbeat which is used to verify and establish a link between the boat and the base and functions as a basic "Hi, I'm still alive message". Since our team was tasked with the 'Entrance and Exit Gates' autonomous challenge, a message for that task was also created. The system was developed using the python programming language and functions as an application.

Physical Setup

On board the robot boat, there are many devices which are single board computers such as raspberry pies and Arduinos. Each single device will contain a software module that is responsible for a particular function for the boat. Such functions include GPS coordinates, Shooting modes, Weather reports, Boat modes, detecting wildlife and detecting different coloured light towers.

These devices communicate which each using the power of the CAN bus (Controller Area Network), which in turn is connected to main robotics controller, the RoboRio. The can bus provides serial communications between devices and they connect together using a single pair of wires or over a CAN hub. CAN is a popular choice for automotive vehicles such as boats as it offers low-level networking solutions for onboard communications. With CAN, The devices can communicate with one another, and via some programming, they can access each other's software modules and gain access to their functions and methods.

The Program

A different NMEA message sentence needs to be defined for the heartbeat and the task for 'Entrance and Exit Gates. For the heartbeat message the message fields include messageID, AEDT Date, AEDT Time, Latitude, N/S, Longitude, E/W indicator, System Mode, UAV status and checksum (Refer to Appendix 1). As the heartbeat message requires data from the boat, drone and other metadata such as date, time a GPS coordinates; it would need to obtain this data from the rest of the system modules. As mentioned above, once all devices are connected by via CAN buses, then the data can be gained from accessor methods located in frc.py (first robotics controller) library. The frc program functions as a central application for all the can devices to connect with and where other program modules can listen

on the Can bus and is where the heartbeat message can extract the statuses of certain modes. The entire heartbeat message fields are placed in variables and concatenated as string data types to form the NVEA sentence and any changes in variable values will be updated dynamically to reflect real-time changes happening in competition. This was basically done by placing the variables in an infinite while-loop and refreshing the heartbeat at a rate of 1Hz/1 second using python time dot sleep method. For the Entrance and Exits gates message, all that is required is obtaining the data from the frc program, placing them in variables and customizing the message fields to the required NVEM sentence format.

TCP Connection

The Technical Director needs to receive the heartbeat message to verify the link in the form of packets. A TCP server/client mode of transmitting data needed to be assembled to make this possible, therefore the python socket module and its inbuilt methods was used as a solution to remedy this task.

The Heartbeat program also functions as a server, and the Technical Director would function as a client as both python programs import the python socket module. The following is representation of the functions and methods used to establish a TCP socket.

THE SERVER:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Creates a socket object defaulted to TCP, which is more reliable than choosing UDP since TCP will retransmit any packets dropped by the network and ensures data is read in the order it was established by the sender.

```
s.bind((host, port))
```

The bind method is used to associate the socket with a specific network interface and port.

```
s.listen(5)
```

This enables the server to accept connections. It makes the server a listening socket. The five in brackets specifies the number of unaccepted connections that the system will allow before refusing new connections.

```
c, addr = s.accept()
```

The. accept() method blocks execution and waits for an incoming connection. When a client (Technical Director) connects, it returns a new socket object representing the connection and a tuple holding the address of the client. The tuple will contain (host, port) for IPv4 connections. It's Important to note that

there is now a new socket object from .accept() since it's the socket that will be used to communicate with the client.

```
c.send(msg.encode)
```

The send method returns the number of bytes sent, which may be less than the size of the data passed in. The msg variable will contain the NVEM sentence format, and encode will format the message into byte type and send the packest through TCP for the requesting client.

THE CLIENT:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

A socket needs to also be created at the client side to enable TCP connection.

```
s.connect((host, port))
```

The client connects to the host IP and port number, the clients need to know IP it wants to connect too and both the server and client need to share the same port number.

```
msg = s.recv(1024)
```

The client will receive the NVEM sentence string from the server in 1024 bytes

```
while msg:
   print(msg.decode())
   msg = s.recv(1024)
```

The client will receive data from the server, decode the bytes into string format. The infinite while loop will let the Technical Director know any changes in the NVEM sentence that would indicate a change of date, time boat or drone status and will always dynamically update.

Potential Improvements

As the author of the Notification system program, the code was written to the limitations of the programming skills of the author. The author can see that a separate Server class could be implemented instead of having the code in the same python file as the heartbeat message. Furthermore, instead of having a separate python file for every single task that requires its own unique NVEM sentence that then needs to be passed to the Technical Director, the author could create one Heartbeat and Messaging System (HMS) python file. The HSM program using conditional statements and obtaining the correct MessageID aka the Protocol Header, could then ascertain that the message that needs to formatted and sent would be an Entrance and Exit Gates Message, a Follow the Path Message, a Scan the Code message or a Follow the Path message. The UAV (drone), AMS (boat) variables were hardcoded as they

were not located in the frc python file, it is unknown whether this data can be obtained via CAN serial as if there were present and accounted for in the frc file, or whether an internal model of the system's state needed to be developed in order to read the packets and then send them to the technical director.

Conclusion

As the author will not be present when the notification system is tested, they cannot ascertain if the program works as intended. It is possible that code at the side of the Technical Director uses a different programming language than python, rendering the program non-functional. The Technical Director also may prefer to function as the server, and the notification system would then become the client, meaning code and logic would need to be reprogrammed.

REFERENCES:

RoboNation, "RobotX Challenge: 2022 Team Handbook," 2 2022. [Online]. Available: https://robonation.org/app/uploads/sites/2/2022/03/202 2-RobotX_Team-Handbook_v2.pdf.

Appendix 1. Heartbeat Message Fields and Entrance and Exit Gates Message

(RoboNation, 2022)

Name	Example	Description	Notes
Message ID	$RXHRB	Protocol Header	
AEDT Date	111221	ddmmyy	Use Australian Eastern Daylight Time (AEDT)
AEDT Time	161229	hhmmss (24hr time format)	Use Australian Eastern Daylight Time (AEDT)
Latitude	21.31198	Decimal degrees	Provides ~1.11m accuracy
N/S indicator	N	N=north, S=South	
Longitude	157.88972	Decimal degrees	Provides ~1.04m accuracy
E/W indicator	W	E=east, W=west	
Team ID	ROBOT	Team ID	5-character code assigned by Technical Director
System Mode	2	Current mode of AMS 1=Remote Operated 2=Autonomous 3=Killed	
UAV Status	1	Current UAV Status 1=Stowed 2=Deployed 3=Faulted	The 'Stowed' state used only when the UAV is secured to the USV. The 'Deployed' state is used whenever the UAV is not on board the USV. The 'Faulted' state is used whenever the UAV is not functioning as designed.
Checksum	11	Bitwise XOR	
<CR><LF>		End of message	

Table 13. RobotX 2022 Heartbeat Message Fields

Heartbeat Example Message: $RXHRB,111221,161229,21.31198,N,157.88972,W,ROBOT,2,1*11

Name	Example	Description
Message ID	$RXGAT	Protocol Header
AEDT date	111221	ddmmyy
AEDT time	161229	hhmmss
Team ID	ROBOT	5-character code assigned by Technical Director
Active Entrance Gate	1	Gate 1, 2, or 3
Active Exit Gate	2	Gate 1, 2, or 3
Checksum	3C	Bitwise XOR
<CR><LF>		End of message

Table 14. Entrance and Exit Gate Message Fields

Entrance and Exit Gate Example Message: $RXGAT,111221,161229,ROBOT,1,2*3C