COEN 320 – Air Traffic Monitoring and Control (ATC)

By

Anna Bui 4021221

Jaskirat kaur 40138320

William Au 40133101

A report submitted to

Professor: Rodolfo Coutinho

Concordia University

9 April 2024

TABLE OF CONTENTS

OBJECTIVES 2
INTRODUCTION 2
ANALYSIS 3
DESIGN 6
IMPLEMENTATION 6
RESULTS (OUTPUT) 14
LESSON LEARNED 18
CONCLUSION 18

OBJECTIVES

The Objective of this project is to implement a air traffic control system that can maintain and order air traffic flow and the distance between aircraft in order to keep a safe environment for safe movement of aircraft - It will display a plan view of the space every few seconds, check all aircraft in the airspace for separation constraint violations at the current time, emit an alarm if a safety violation is found, store the airspace in a history file every seconds, as well as the operators requests and commands in a log file.

INTRODUCTION

Air Traffic Monitoring and Control (ATC) is a complex real-time system composed of numerous subsystems. It is an important component of aviation safety and efficiency. Its main priority is to ensure the orderly movement of aircraft within a designated airspace which prevents collisions and ensures a safe navigation of each aircraft. Also, there are tower controllers that manage the airspace traffic, including sending alerts and commands to specific aircraft. In the context of this course, the simplified ATC controls aircraft flows in the enroute control area. The en-route control will handle each aircraft movement from the moment they are in the en route ATC, and will be controlled by a TRACON neighbor site from the moment they leave the en route control airspace. Since the ATC is composed of numerous subsystems that all communicate with each other, it is assumed that the implementation will utilize multithreading, server-clients, and a timer system to ensure real time operation and response to incoming commands. The threading will allow the concurrent execution of various tasks within our ATC system, while the timer will ensure updates and responses to aircraft movements and other events (alerts, commands, etc.).

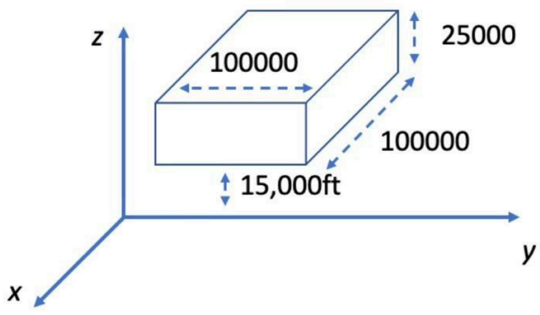


Figure 1: 3D control airspace

ANALYSIS

As mentioned above, the simplified ATC is composed of 5 subsystems, where each communicates among them by sending and receiving real time information.

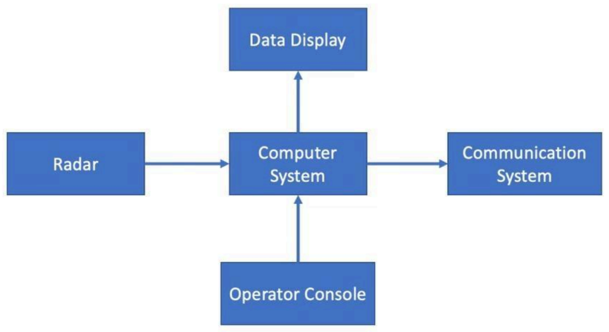


Figure 2: Components of the simplified ATC

AIRCRAFT

Each aircraft is implemented as a process that can update their location with their speed and previous location; Hence, in our ATC, the class should be implemented to to send the radar information,. such as their ID, position, and speed, upon requests.

RADAR

The main function of the radar class is to receive and process information from both primary surveillance radar and secondary surveillance radar systems. In fact, the information received will include details about multiple aircraft that are within the airspace; such as their flight identification, their flight level, their speed, and their position in the xyz axis. Furthermore, the radar class is responsible for handling all communication between the radar system and the computer system as it continuously transmits aircraft data in real time.

In our simplified ATC, the radar class will know the location of each aircraft in the monitored space by communicating directly with the process of each aircraft.

In fact, once the program starts, it will read an input file, which the radar will process and then transmit via a client-server implementation to the computer system, where it will become available for further function of our simplified ATC. In addition, the airspace will be stored in a log history file every 30 seconds.

The input file should contain information structured as Time, ID, X, Y, Z, SpeedX, SpeedY, and SpeedZ, where Time is the moment the aircraft appears in the bounded area, ID is the identification of the aircraft, (x,y,z) are the coordinates of the entering aircraft at the boundaries of the area, (vx,vy,vz) are the speeds of the aircraft in each coordinate dimension.

COMPUTER SYSTEM

On the other hand, the computer system is responsible for periodically (current time + n, where the controller can choose the n period) computing the distance between different aircraft in the airspace (with the data received from the radar) to determine if there is any violation, such as the separation constraint. The computer system class is hence responsible for sending alerts and notifications within 3 minutes to the operator if a violation is bound to happen. In addition, it also sends the data display class the ID and position of the aircraft that will be shown on the screen of the controller. Therefore, the computer system class plays a crucial role as it is the heart of all subsystems that support the overall air traffic control infrastructure.

In our simplified ATC, the computer system server will store all real-time aircraft data (client-server communication with RADAR) into a vector and constantly compute the distance between them to ensure that no collision is detected. However, if there is a violation, it will emit an alert on the screen to notify the operator (client-server with operator console). It will also send the collected real-time data to the data display class, including the ID and position of the aircraft shown on the screen of the controller (client-server with data display).

DATA DISPLAY

The data display processes the aircraft data sent from the computer system and displays its real-time position onto a 2D printout of the airspace every 5 seconds.

OPERATOR CONSOLE

The operator console will allow the controller to send commands to aircraft within the airspace. The different commands that can be sent is to change the speed, altitude, or position of a specified aircraft. It can also be used by the controller to request information about a specific aircraft and ask it to be shown in the data display.

In our simplified ATC, the operator console will prompt the operator to a command, and via a client-server architecture, it will be sent back to the computer system, in order to allow the communication system to fetch the command and act upon it. In addition, all operator requests and commands are stored in a log file.

COMMUNICATION SYSTEM

Finally, the communication system is responsible for transmitting the commands received from the controller to the specified aircraft - where the aircraft can be notified for any change.

In our simplified ATC, the communication system will receive, via the client-server architecture, all commands sent from the operator console to the computer system. From there, the class will fetch the commands directly from the computer system, and act upon the command (such as changing the positions, speed, altitude, etc) by modifying/reading the values of the original input file containing all aircraft.

DESIGN

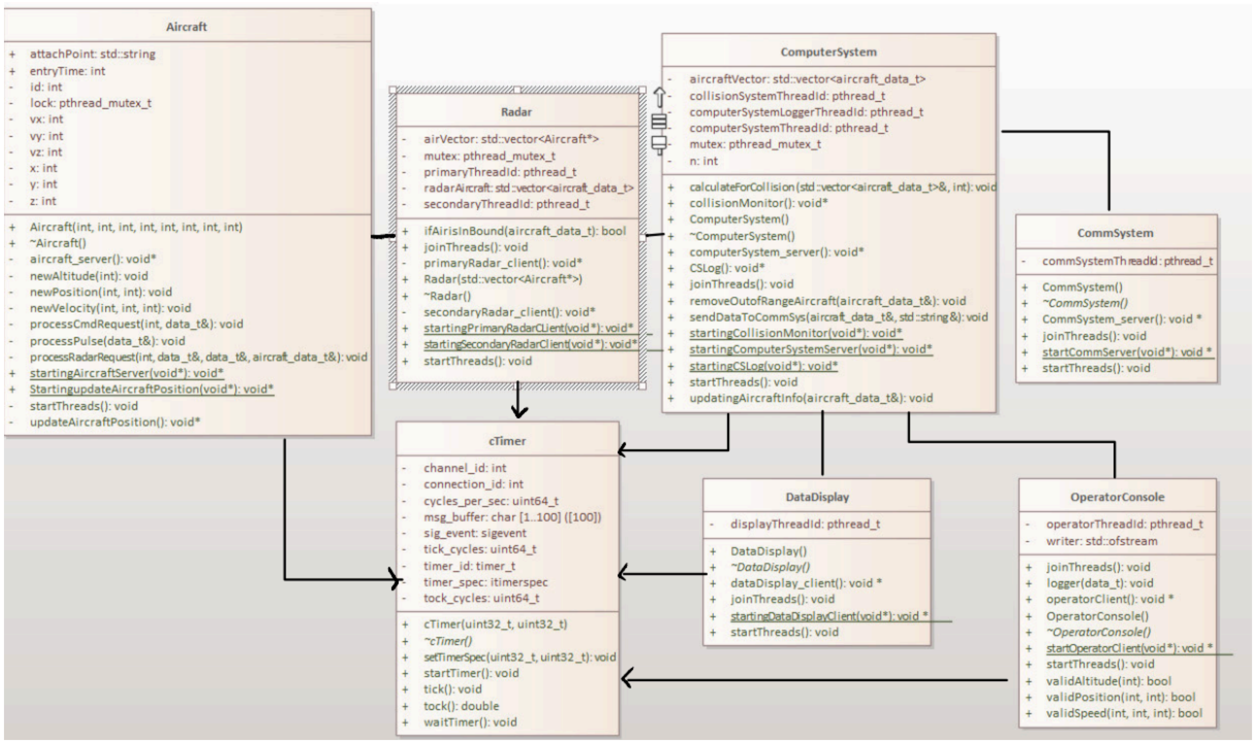

Aircraft
+ attachPoint: std::string
+ entryTime: int
- id: int
- lock: pthread_mutex_t
- vx: int
- vy: int
- vz: int
- x: int
- y: int
- z: int
+ Aircraft(int, int, int, int, int, int, int, int)
+ ~Aircraft()
- aircraft_server(): void*
- newAltitude(int): void
- newPosition(int, int): void
- newVelocity(int, int, int): void
- processCmdRequest(int, data_t&): void
- processPulse(data_t&): void
- processRadarRequest(int, data_t&, data_t&, aircraft_data_t&): void
+ startingAircraftServer(void*): void*
+ StartingupdateAircraftPosition(void*): void*
- startThreads(): void
- updateAircraftPosition(): void*
Radar
- airVector: std::vector<Aircraft*>
- mutex: pthread_mutex_t
- primaryThreadId: pthread_t
- radarAircraft: std::vector<aircraft_data_t>
- secondaryThreadId: pthread_t
+ ifAirisInBound(aircraft_data_t): bool
+ joinThreads(): void
- primaryRadar_client(): void*
+ Radar(std::vector<Aircraft*>)
+ ~Radar()
- secondaryRadar_client(): void*
+ startingPrimaryRadarCLient(void*): void*
+ startingSecondaryRadarClient(void*): void*
+ startThreads(): void
ComputerSystem
- aircraftVector: std::vector<aircraft_data_t>
- collisionSystemThreadId: pthread_t
- computerSystemLoggerThreadId: pthread_t
- computerSystemThreadId: pthread_t
- mutex: pthread_mutex_t
- n: int
+ calculateForCollision(std::vector<aircraft_data_t>&, int): void
+ collisionMonitor(): void*
+ ComputerSystem()
+ ~ComputerSystem()
+ computerSystem_server(): void*
+ CSLog(): void*
+ joinThreads(): void
+ removeOutofRangeAircraft(aircraft_data_t&): void
+ sendDataToCommSys(aircraft_data_t&, std::string&): void
+ startingCollisionMonitor(void*): void*
+ startingComputerSystemServer(void*): void*
+ startingCSLog(void*): void*
+ startThreads(): void
+ updatingAircraftInfo(aircraft_data_t&): void
CommSystem
- commSystemThreadId: pthread_t
+ CommSystem()
+ ~CommSystem()
+ CommSystem_server(): void *
+ joinThreads(): void
+ startCommServer(void*): void *
+ startThreads(): void
cTimer
- channel_id: int
- connection_id: int
- cycles_per_sec: uint64_t
- msg_buffer: char [1..100] ([100])
- sig_event: sigevent
- tick_cycles: uint64_t
- timer_id: timer_t
- timer_spec: itimerspec
- tock_cycles: uint64_t
+ cTimer(uint32_t, uint32_t)
+ ~cTimer()
+ setTimerSpec(uint32_t, uint32_t): void
+ startTimer(): void
+ tick(): void
+ tock(): double
+ waitTimer(): void
DataDisplay
- displayThreadId: pthread_t
+ DataDisplay()
+ ~DataDisplay()
+ dataDisplay_client(): void *
+ joinThreads(): void
+ startingDataDisplayClient(void*): void *
+ startThreads(): void
OperatorConsole
- operatorThreadId: pthread_t
- writer: std::ofstream
+ joinThreads(): void
+ logger(data_t): void
+ operatorClient(): void *
+ OperatorConsole()
+ ~OperatorConsole()
+ startOperatorClient(void*): void *
+ startThreads(): void
+ validAltitude(int): bool
+ validPosition(int, int): bool
+ validSpeed(int, int, int): bool

Figure 3 : UML Graph

IMPLEMENTATION

cTimer Class

The cTimer class is used to ensure the real-time operation of the system.

cTimer(uint32_t sec, uint32_t msec): It has a constructor that initializes the object with a specified duration in seconds and milliseconds. To do so, it creates a channel and initializes it using the real time clock.

In order to run the ATC with multiple subsystem - multiple functions involving the timer were implemented, such as **void startTimer(), void setTimerSpec(uint32_t sec, uint32_t nano), void waitTimer(), void tick(), -** used to record the current value of the clock cycle, and **double tock(),** which was used to calculate the time elapsed between the tick and tock and return it.

<u>Structure.h</u>

This header class is used to define the different types of data that our ATC can process as well as the different channels used between the subsystems to communicate with each other. The two channels COMP_SYS_ATTACH, and COMM_SYS_ATTACH we're defined as the system's attachment point for the computer system (the heart of the system), and for the communication system to handle the modifications/requests from the operator. As for the structures, we've defined the following:

msg_header_t: This represents the different headers or each message; this will allow the system to recognize the type of data that the system will receive/send.

aircraft_data_t: This represents the aircraft data and contains an aircraft information, such as its entrytime, id, position (x,y,z) and speed (vx,vy,vz) , a attachPoint (Since aircraft will be multiple thread and receiving commands from the operator respectively), and a bool that ensures that the aircraft is still in the bounded airspace.

display_data_t: This stores all of the aircraft_data t aircrafts in a vector in order to use their information to display them in a plane. In addition, it has a variable that keeps track of the specific aircraft the operator might request for augmented information.

command_data_t: This is used to decode the operator's command; it store the requested aircraft information in the structure of aircraft_data, the string of command requested, a bool that verifies that the aircraft requested is existing, and a variable to store the new period (for crash verification) requested by operator.

data_t: This struct bundles all of the various types of data together.

Aircraft Class

The aircraft class is implemented to simulate an aircraft in the airspace that can also receive requests to change its current information;

Aircraft(int time, int id, int x, int y, int z, int vx, int vy, int vz): This constructor initialize the aircraft with the following parameters, such as the entry time, the craft id, the position in the x,y,z axis and velocity (vx,vy,vz). It also initializes a mutex to avoid any race conditions. It also constructs an attach point string used to name the channel that will be used to communicate with other systems in order to update, if requested by the operator, new values.

There’s also a destructor that will destroy the mutex to free the ressources.

To avoid any race conditions as multiple thread will be handling the values; mutex were used in the updates functions:

void newPosition(int newX, int newY): This function updates the current position x and y to a new one
void newAltitude(int newAltitude): This function updates the current position z to a new one
void newVelocity(int newVX, int newVY, int newVZ):

void * updateAircraftPosition(): This function continuously updates the aircraft’s position in a loop based on its velocity; which simulates the movement of the aircraft. With the help of a timer set with a period of 1s(cTimer object), the position will be updated at a fix interval.

void* aircraft_server(): This server function handles the incoming messages from the radars and command requests sent from the operator. It creates a channel to process different type of messages that are recognized by their headers (defined in the defs.h) and acts upon it;

For instance, the PRIMARY_RADAR header fills in the aircraft data information (of aircraft_data_t type); if its the SECONDARY_RADAR, it will process the requests by constructing a response with the current state of the craft and its positions and velocity with the **void processRadarRequest(int rcvid, const data_t& msg, data_t& request_msg, aircraft_data_t& airData)** function; if it's the COMMAND, it will run a function that processes the command request sent from the operator; which will call the corresponding functions to update the position/altitude.

To ensure that all messages are well received and connected the **void processPulse(const data_t& msg)** function will process the system pulses.

void* startingAircraftServer(void* context), and void* StartingupdateAircraftPosition(void* context): Those are static wrappers used to start threads, so they can run separately.

Radar Class

The Radar class is implemented to keep watch of the current airplanes present in the bounded space;

Radar::Radar(vector<Aircraft*> aircraftVector) : It has a constructor that initializes the radar object with a list of aircraft; it also sets up a mutex to ensure that it runs safely ensuring that the shared data are not prone to any race conditions.

There's also a destructor that will destroy the mutex to free the ressources.

bool ifAircraftisInBound(aircraft_data_t aircraft): This bool function is implemented to verify that the aircraft positions are indeed corresponding to the control airspace in figure 1; Hence it verifies if the aircraft is in between the bounds and returns a true/false value depending on the results.

void* primaryRadar_client(): This function is used to represent the primary radar that is constantly monitoring the airspace for all aircraft that are within the range. In fact, it iterates over a list of aircraft and requests for each of their positions and uses the **bool ifAirisInBound(aircraft_data_t aircraft)** function to verify that they are within the set bounds. At the same time, it communicates through a channel with the computer system (COMP_SYS_ATTACH) to update it of the aircrafts positions by storing them in the aircraft_data_t struct, and label it (header) as a COMP_PRIMARY.. It also uses a Timer to regulate the frequency of the radar scans (cTimer object); in this case we've set the scans for a period of 1s.

void* secondaryRadar_client(): This function is similarly implemented to the primaryRadar_client(); In fact, the secondary Radar is used to communicate directly with a specific aircraft. Through the channel COMM_SYS_ATTACH, it creates a connection (different for each plane) and requests information of the craft by labeling them as a SECONDARY_RADAR. It also sends, with the label COMP_SECONDARY, the requested data to the computer system. It also uses a Timer to regulate the frequency of the radar scans (cTimer object); in this case we've set the scans for a period of 1s.

The client functions both uses mutexes to protect shared data, such as the radarAircraft list to ensure there is no race conditions when multiple threads are accessing it, They also both support Inter-Process communication that can handle if the messages are well sent; if not, error messages will be displayed to the console.

void* startingPrimaryRadarCLient(void* context) and void* startingSecondaryRadarClient(void* context): are the Wrapper functions that will allow both Primary and Secondary radar client to run in separate threads

Computer System Class

The computer System class is the heard of all of our subsystem; it will send and process different types of data throughout the existing systems:

ComputerSystem(): The constructor initializes the object by setting the default time for the cr the future collision predictions to 180s (3 minutes). It also initializes a mutex to ensure a thread safe environment for the shared resources throughout the functions.
There’s also a destructor that will destroy the mutex to free the ressources.

void updatingAircraftInfo(const aircraft_data_t& airData): This function updates the information for an existing aircraft and constantly adds it to a vector list to track it; To avoid having copies, it checks if the aircraft exists; it uses a bool integer that is set to true when the for loop iterates through all the lists and finds the aircraft. Only when it doesn’t exist; it will be added to the list. If it already exists; it just updates the information. To ensure the safe handling of the list; a mutex is used.

void removeOutofRangeAircraft(const aircraft_data_t& airData): This function removes an aircraft of the vector list system if it is out of bounds. To ensure the safe handling; mutex is used.

void* collisionMonitor() : This function continuously monitors all the aircraft (in the vector list) for potential collisions based on each of their positions and velocities. To do so, it uses a temp valur to store a copy of the current aircraft information to predict its future positions and checks for any potential collision with the function **calculateForCollision(const vector<aircraft_data_t>& tempVector, int time).** Since the list of vector is once again accessed, mutex were added to ensure no race conditions. It also uses a timer to regulate the frequency of the monitoring; in our case the period was set to 1 (cTimer object)

void calculateForCollision(const vector<aircraft_data_t>& tempVector, int time) : This function is an algorithm that calculates the distance between two aircrafts. Hence, it iterates through the list and verifies, for all x,y,z positions, if they are within a certain distance of each other. If the distance is too close; it will emit a collision warning with the specific time.

void *computerSystem_server(): This function is the server of the computer system; it will handle various types of messages from the radar, operator, and data display, by updating aircraft info; or removing out of range aircraft, and responding to any request from the data display. In fact, it uses QNX functions for the message passing and channel creation COMP_SYS_ATTACH to communicate with the other system - it will also send error messages to the console if messages are not properly sent or received. To differentiate the messages going in and out the channel; they are labeled according to the headers defined in the defs.h class.

void sendDataToCommSys(const aircraft_data_t& airData, const std::string& cmd): This function is used to send data to the communication system; which are the commands or information updates requested from the operator. It communicates through the channel COMM_SYS_ATTACH. With the message passing method; it sends information and ensures that it is sent through; or else an error message is displayed.

void* CSLog(): This function periodically logs the current status of all tracked aircraft in the vector list into a file stored in the /data/home/qnxuser/systemLog.txt directory. The timer is set to a period of 30 seconds and will ensure that the system is logged every 30 seconds of all the current aircraft in the list vector of aircraft bounded. Again, since the list is used, mutex is used to avoid any race conditions.

void* startingCSLog(void* context) , void * startingComputerSystemServer(void * context), and void* startingCollisionMonitor(void* context) : Finally, those are all the wrappers function used to allow each to run as separate threads.

- DisplayData Class

The DisplayData class is used to collect the information from the computer system and display it in a visual 2D plane for the user

void* dataDisplay_client(): This client function requests for information of aircrafts from the computer system throughout the COMP_SYS_ATTACH channel and displays the data received. To ensure that the connection is properly established and that messages were correctly retrieved; QNX functions were used (if unsuccessful, error messages are displayed to the console). The airspace is represented by a 25x25 grid and is initialized to -1 for empty spaces. The grid will be implemented according to the cell size; in our case we assumed that a cell represents a 5000x5000 unit area. A Timer object was used to ensure the display would refresh every 5 seconds. The while loop will support the function for requests of data by sending a message with the header type DATA_DISPLAY (defined in the defs.h). Hence, once the client receives a response, it iterates through the list of current aircrafts received and resets the grids according to their x and y positions. To facilitate the reading in the grid, the aircraft will be represented by their id's in the grid. The loop will then wait for the existing timer (5s) to expire before requesting new information from the computer system. However, if an error occurs the program exits the loop and closes the connection.

void* startingDataDisplayClient(void* context): This is the Wrapper function that will allow the client to run in a separate thread

- Operator Console Class

This Operator Console Class is used by the operator to send specific commands to the computer system; such as requesting and changing current aircraft information

void* operatorClient(): This function acts as the client - It will communicate with the computer system server through the channel COMP_SYS_ATTACH and message passing methods to verify that the message is sent and received correctly. In fact, it enters a while loop where it continuously accepts commands from the operator. In other words; it constantly prompts the operator for a specific command such as

1-Request augmented info
2-Change speed
3-Change position
4-Change altitude
5-Change collision prediction time

To ensure correct that the operator inputs are correct; multiple functions were used to verify if the input is indeed an integer; if it corresponds to the list of command, or if its following the constraints (functions in the below are called for the verification), It will also update the log to all commands imputed by the operator (also explained in the below).

void logger(data_t msg): This function stores all the commands requested from the operator to a log file mentioning the type of commands and the aircraft current/new information, or the new updated prediction times.To differentiate the type of command; it verifies the header of each request (defined in the defs.h)

bool validSpeed(int newSpeedX, int newSpeedY, int newSpeedZ): This function ensures that the input for all speed components are positive.
bool validPosition(int x, int y: This function ensures that the input for the positions are respective of the defined bounded from figure 1.
bool OperatorConsole::validAltitude(int z): This function ensures that the input for the altitude is within the range from figure 1.

void* startOperatorClient(void* context): This is the Wrapper function that will allow the client to run in a separate thread

- Communication System Class

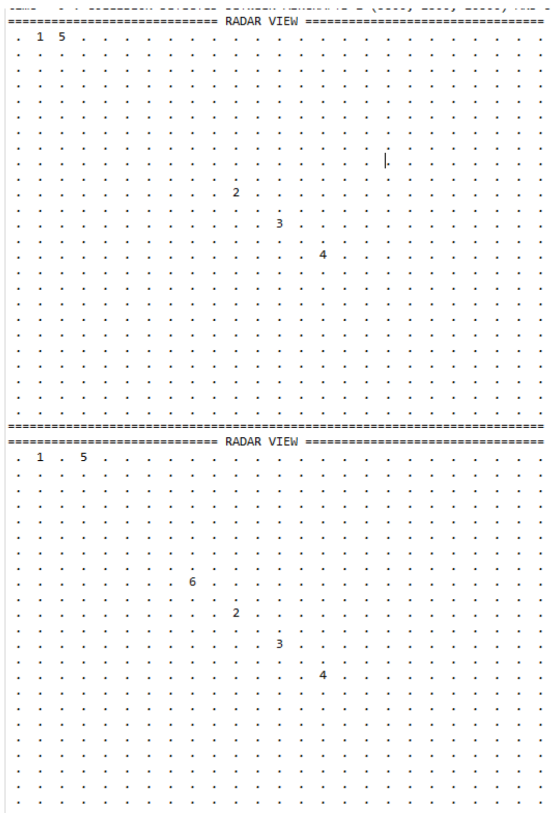
The communication system class is implemented to transmit the commands to the specific aircraft ; where it will be notified.

void* CommSystem_server(): This server function establishes a communication channel to receive messages and commands intended for specific aircraft. It uses the channel COMM_SYS_ATTACH. It has a loop that consistantly waits for a message; and once a message is received, it checks the message types with the help of the headers name (defined in defs.h) For each command; it opens a connection with the target aircrafts communication channel and forwards it the command. Once completed, it closes the information. The timer is used to prevent overload, which ensures that the server is not overwhelmed with messages, Throughout the loop; it also has QNX functions that verifies the pulse, if the connection is successful (if not sends an error message), or the connection responses are established(if not sends an error message).

void* startCommServer(void* context): This is the Wrapper function that will allow the server to run in a separate thread,

RESULTS (OUTPUT)

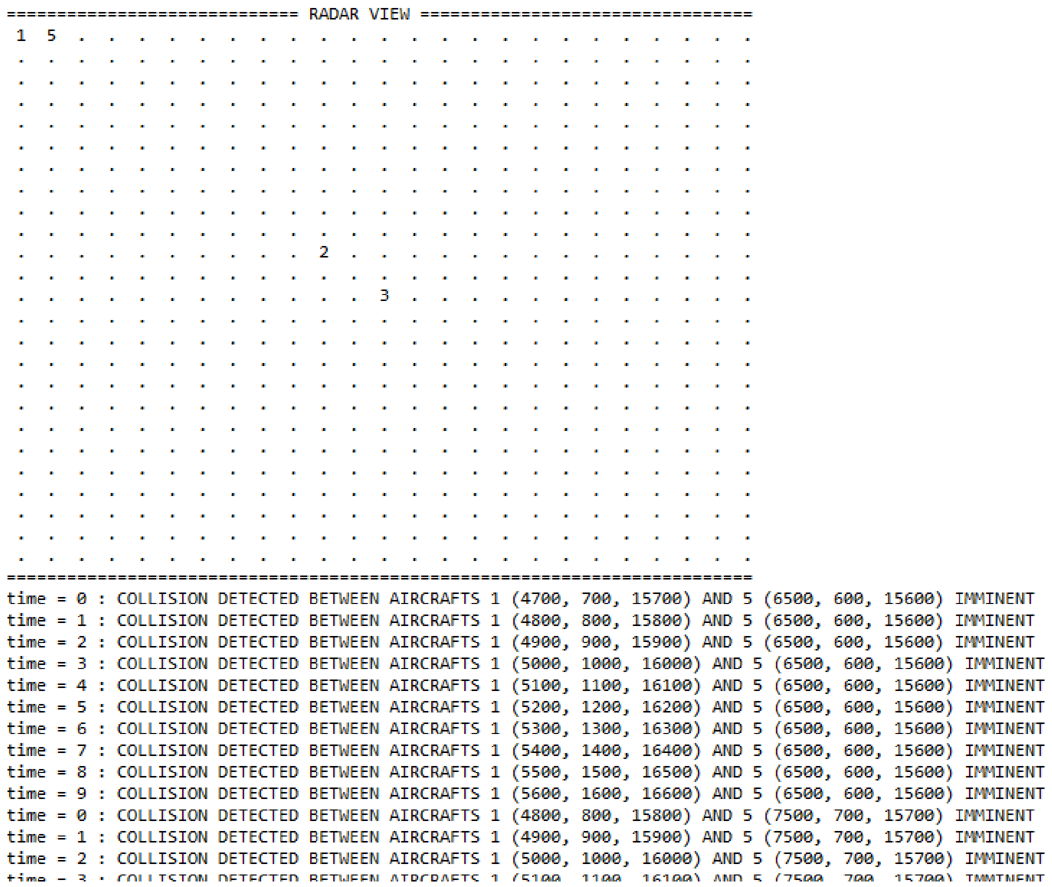
a) Data Display every 5 seconds



```
============================ RADAR VIEW ================================
 .  1  5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  2  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  3  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  4  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
=========================================================================
============================ RADAR VIEW ================================
 .  1  .  5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  6  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  2  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  3  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  4  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

As expected, the data display has a refresh rate of approximately 5 seconds and displays the current airspace to the user. Each number represents the aircraft ID.

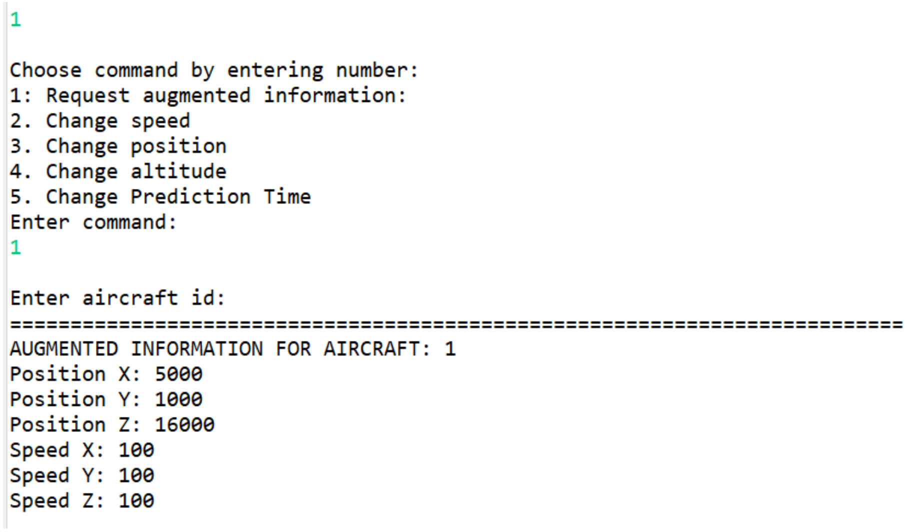
b) Collision Detection



```
============================== RADAR VIEW ================================
 1  5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  2  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  3  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
==========================================================================
time = 0 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4700, 700, 15700) AND 5 (6500, 600, 15600) IMMINENT
time = 1 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4800, 800, 15800) AND 5 (6500, 600, 15600) IMMINENT
time = 2 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4900, 900, 15900) AND 5 (6500, 600, 15600) IMMINENT
time = 3 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5000, 1000, 16000) AND 5 (6500, 600, 15600) IMMINENT
time = 4 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5100, 1100, 16100) AND 5 (6500, 600, 15600) IMMINENT
time = 5 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5200, 1200, 16200) AND 5 (6500, 600, 15600) IMMINENT
time = 6 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5300, 1300, 16300) AND 5 (6500, 600, 15600) IMMINENT
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (6500, 600, 15600) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (6500, 600, 15600) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5600, 1600, 16600) AND 5 (6500, 600, 15600) IMMINENT
time = 0 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4800, 800, 15800) AND 5 (7500, 700, 15700) IMMINENT
time = 1 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4900, 900, 15900) AND 5 (7500, 700, 15700) IMMINENT
time = 2 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5000, 1000, 16000) AND 5 (7500, 700, 15700) IMMINENT
time = 3 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5100, 1100, 16100) AND 5 (7500, 700, 15700) IMMINENT
```

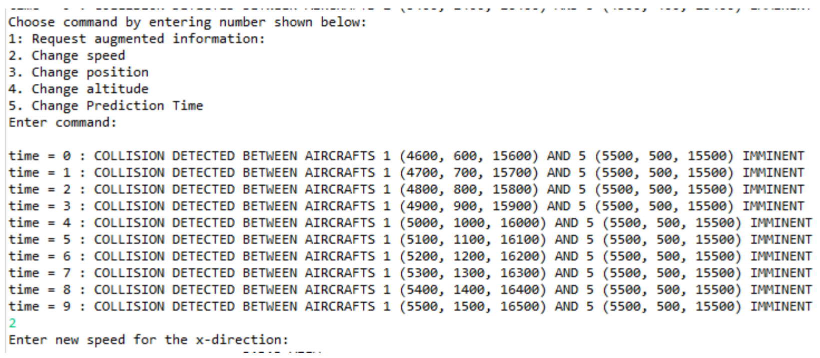
c) Operator Controls

i) 1-Request information for craft

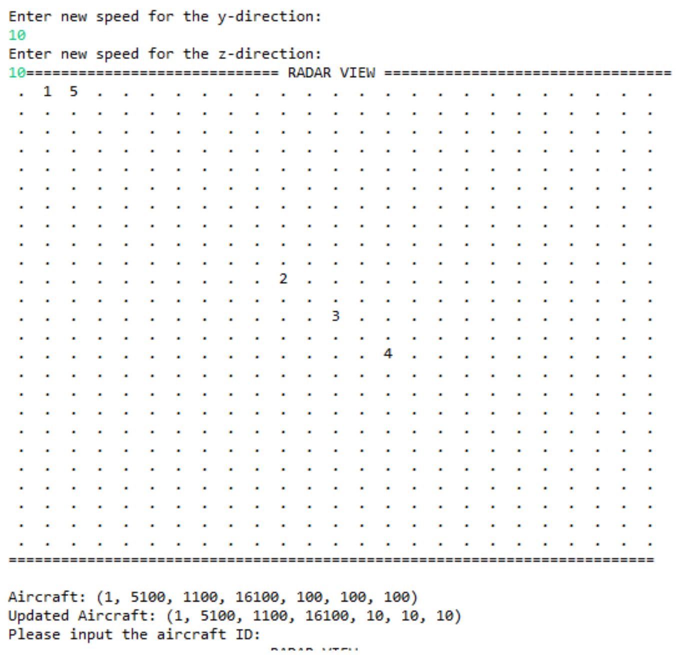


```
1

Choose command by entering number:
1: Request augmented information:
2. Change speed
3. Change position
4. Change altitude
5. Change Prediction Time
Enter command:
1

Enter aircraft id:
==========================================================================
AUGMENTED INFORMATION FOR AIRCRAFT: 1
Position X: 5000
Position Y: 1000
Position Z: 16000
Speed X: 100
Speed Y: 100
Speed Z: 100
```

ii) 2-Change velocity(VX,VY,VZ)



```
Choose command by entering number shown below:
1: Request augmented information:
2. Change speed
3. Change position
4. Change altitude
5. Change Prediction Time
Enter command:

time = 0 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4600, 600, 15600) AND 5 (5500, 500, 15500) IMMINENT
time = 1 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4700, 700, 15700) AND 5 (5500, 500, 15500) IMMINENT
time = 2 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4800, 800, 15800) AND 5 (5500, 500, 15500) IMMINENT
time = 3 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4900, 900, 15900) AND 5 (5500, 500, 15500) IMMINENT
time = 4 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5000, 1000, 16000) AND 5 (5500, 500, 15500) IMMINENT
time = 5 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5100, 1100, 16100) AND 5 (5500, 500, 15500) IMMINENT
time = 6 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5200, 1200, 16200) AND 5 (5500, 500, 15500) IMMINENT
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5300, 1300, 16300) AND 5 (5500, 500, 15500) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (5500, 500, 15500) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (5500, 500, 15500) IMMINENT
2
Enter new speed for the x-direction:
```

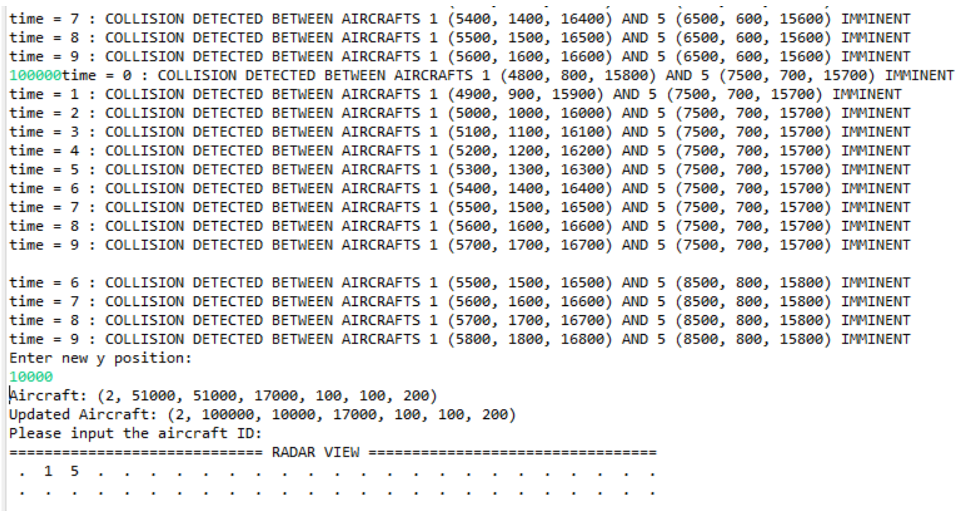


```
Enter new speed for the y-direction:
10
Enter new speed for the z-direction:
10============================ RADAR VIEW ================================
 .  1  5  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  2  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  3  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  4  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
==========================================================================

Aircraft: (1, 5100, 1100, 16100, 100, 100, 100)
Updated Aircraft: (1, 5100, 1100, 16100, 10, 10, 10)
Please input the aircraft ID:
```

As expected, when the user inputs the aircraft ID, it prompts it to choose a command; if 2 is selected - user will be prompted to input a new speed value. Hence, modified value is now saved for the specified aircraft ID

iii) 3-Change Position (X,Y)

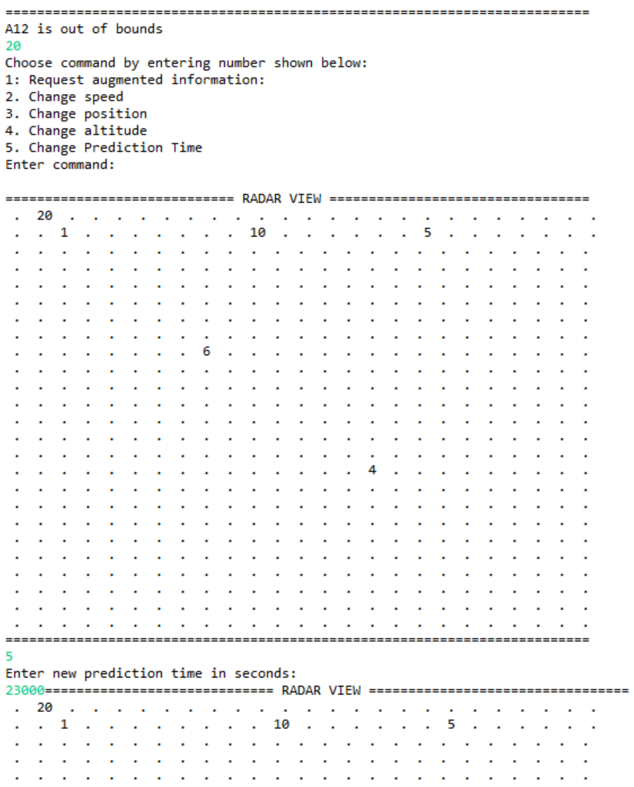
Similarly, the user is prompt to input the new x,y position if 3 is selected - Hence as expected , modified value is now saved for the specified aircraft ID



```
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (6500, 600, 15600) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (6500, 600, 15600) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5600, 1600, 16600) AND 5 (6500, 600, 15600) IMMINENT
100000time = 0 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4800, 800, 15800) AND 5 (7500, 700, 15700) IMMINENT
time = 1 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4900, 900, 15900) AND 5 (7500, 700, 15700) IMMINENT
time = 2 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5000, 1000, 16000) AND 5 (7500, 700, 15700) IMMINENT
time = 3 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5100, 1100, 16100) AND 5 (7500, 700, 15700) IMMINENT
time = 4 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5200, 1200, 16200) AND 5 (7500, 700, 15700) IMMINENT
time = 5 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5300, 1300, 16300) AND 5 (7500, 700, 15700) IMMINENT
time = 6 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (7500, 700, 15700) IMMINENT
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (7500, 700, 15700) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5600, 1600, 16600) AND 5 (7500, 700, 15700) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5700, 1700, 16700) AND 5 (7500, 700, 15700) IMMINENT

time = 6 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (8500, 800, 15800) IMMINENT
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5600, 1600, 16600) AND 5 (8500, 800, 15800) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5700, 1700, 16700) AND 5 (8500, 800, 15800) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5800, 1800, 16800) AND 5 (8500, 800, 15800) IMMINENT
Enter new y position:
10000
Aircraft: (2, 51000, 51000, 17000, 100, 100, 200)
Updated Aircraft: (2, 100000, 10000, 17000, 100, 100, 200)
Please input the aircraft ID:
============================ RADAR VIEW ================================
 . 1 5 . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
```

iv) 4-Change Altitude (Z)

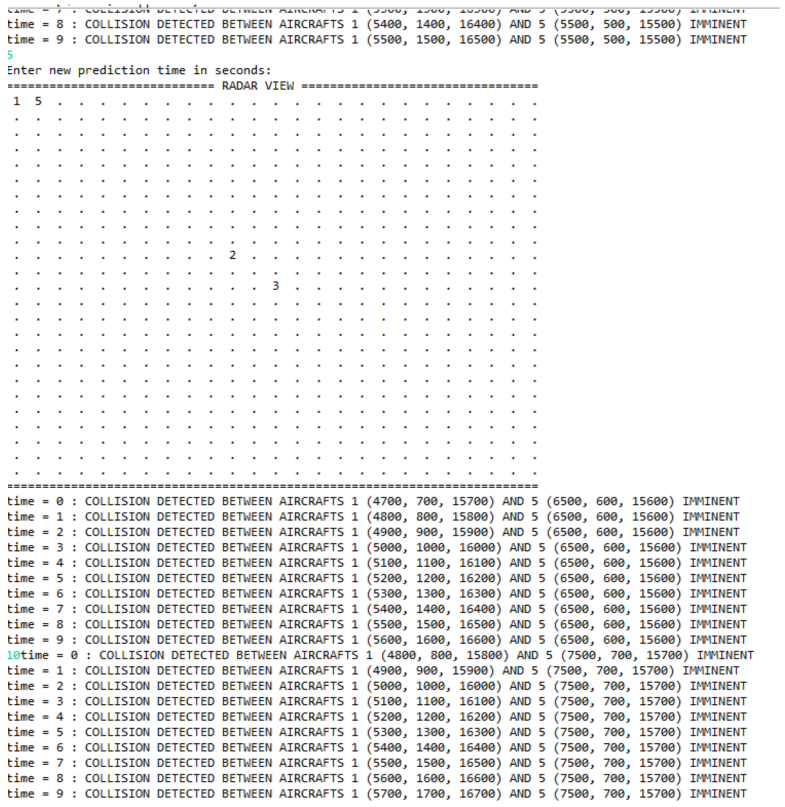
Similarly, the user is prompt to input the new x,y position if 3 is selected - Hence as expected , modified value is now saved for the specified aircraft ID



```
==========================================================================
A12 is out of bounds
20
Choose command by entering number shown below:
1: Request augmented information:
2. Change speed
3. Change position
4. Change altitude
5. Change Prediction Time
Enter command:

============================ RADAR VIEW ================================
 . 20 . . . . . . . . . . . . . . . . . . . . . . .
 . . 1 . . . . . . . 10 . . . . . . 5 . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . 6 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . 4 . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
==========================================================================
5
Enter new prediction time in seconds:
23000============================ RADAR VIEW ================================
 . 20 . . . . . . . . . . . . . . . . . . . . . . .
 . . 1 . . . . . . . . 10 . . . . . . 5 . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
```

v) 5-Change Prediction Time

As we can see, the user when selecting 5 is prompt to change the prediction time; in this case the user changed it to 10. Hence we can see that it work as expected as the frequency of the collision is set to 10s



```
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5300, 1300, 16300) AND 5 (5500, 500, 15500) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (5500, 500, 15500) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (5500, 500, 15500) IMMINENT
5
Enter new prediction time in seconds:
============================ RADAR VIEW ================================
 1 5 . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . 2 . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . 3 . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . .
========================================================================
time = 0 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4700, 700, 15700) AND 5 (6500, 600, 15600) IMMINENT
time = 1 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4800, 800, 15800) AND 5 (6500, 600, 15600) IMMINENT
time = 2 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4900, 900, 15900) AND 5 (6500, 600, 15600) IMMINENT
time = 3 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5000, 1000, 16000) AND 5 (6500, 600, 15600) IMMINENT
time = 4 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5100, 1100, 16100) AND 5 (6500, 600, 15600) IMMINENT
time = 5 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5200, 1200, 16200) AND 5 (6500, 600, 15600) IMMINENT
time = 6 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5300, 1300, 16300) AND 5 (6500, 600, 15600) IMMINENT
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (6500, 600, 15600) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (6500, 600, 15600) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5600, 1600, 16600) AND 5 (6500, 600, 15600) IMMINENT
10time = 0 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4800, 800, 15800) AND 5 (7500, 700, 15700) IMMINENT
time = 1 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (4900, 900, 15900) AND 5 (7500, 700, 15700) IMMINENT
time = 2 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5000, 1000, 16000) AND 5 (7500, 700, 15700) IMMINENT
time = 3 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5100, 1100, 16100) AND 5 (7500, 700, 15700) IMMINENT
time = 4 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5200, 1200, 16200) AND 5 (7500, 700, 15700) IMMINENT
time = 5 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5300, 1300, 16300) AND 5 (7500, 700, 15700) IMMINENT
time = 6 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5400, 1400, 16400) AND 5 (7500, 700, 15700) IMMINENT
time = 7 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5500, 1500, 16500) AND 5 (7500, 700, 15700) IMMINENT
time = 8 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5600, 1600, 16600) AND 5 (7500, 700, 15700) IMMINENT
time = 9 : COLLISION DETECTED BETWEEN AIRCRAFTS 1 (5700, 1700, 16700) AND 5 (7500, 700, 15700) IMMINENT
```

LESSON LEARNED

This project has introduced us to real-time C++ coding using QNX Momentics, as well as the implementation of threads and client-server communication channels - where our understanding of system resource management, and task scheduling was used. In addition, working with threads allowed us to effectively manage concurrent execution of tasks within a real-time system, while also having to avoid race conditions and deadlocks. The client-server communication using channels was also another important aspect of our ATC project, we learned how to establish communication channels between different subsystems, allowing them to exchange data in a real-time environment

CONCLUSION

In conclusion, the team was able to implement all of the requirements necessary for the simplified ATC. The program can successfully run with multiple threads without meeting any race conditions as mutex was carefully used at danger areas. In addition, all subsystems can communicate with each other by requesting/sending data throughout the respective channel and process each message according to their header type. We also face challenges in simulating real time scenarios and getting familiar with the QNX functions to verify message passing, etc. For future work, we could implement a more accurate algorithm for collision detection to improve the safety aspect of the simulation; or we could develop a more intuitive operator console and data display to provide the user a better overview of the simulation.