Software System in Hyperloop Pod

Ruslan Nikolaev
Waterloo University
Waterloo, Canada

Dinara Nikolaeva
National Research University
Higher School of Economics,
Moscow, Russia

Rinat Idiyatullin
Kazan State Power
Engineering University,
Kazan, Russia

***Abstract*—The Hyperloop is high-speed ground-based transportation system concept; a supersonic train line stretching across the country in airless tubes, which stands as competition for air, train, and car transportation for distances of travel from 200 to 1100 km. This paper concentrates on software - applied in the Hyperloop pod called Goose 3 [1], which was developed by the team from University of Waterloo [2] - Waterloop [3]. This paper is concentrated on roadblocks, that were faced during the design and development process; to be more specific - building a reliable and scalable infrastructure that allows for complete control of the pod throughout the launch. In the pod, electronics and software play a crucial role, as traveling at high velocities requires immediate response to real-time vehicle conditions obtained from sensors on-board, which raises it's own unique challenges. This paper looks at the work completed by team Waterloop in areas of design of the electrical and software system.**

I. Introduction

A. SpaceX competition

In this paper we look upon software problems, as well as, we introduce problems, which were faced by team Waterloop and theorize about our worries and possible solutions for future work. In 2015-2018 SpaceX has been holding annual student competition in order to advance research of the proposed idea, as well as test it in different complications. Testing tube is 1.6 km long. Participating teams are not allowed to modify the tube in any way [4].

The competition is judged based on two main criteria: the highest speed achieved by team's pod and completion of the entire test track; i.e. the pod has traveled the length of the entire tube. During the competition, the first week is spent directly with SpaceX engineers verifying the pod design and testing every pod sub-system. The weekend is used for the competition race.

The competition is open for any applications of propulsion, from friction wheels to linear induction motor, to get up to speed. Our academic research into the Hyperloop concept has focused mostly on system integration of electrical and software engineering. The tools, sketches and code is available under MIT license [5].

As part of the SpaceX competition, team Waterloop had developed a complete solution for the pod as well as the entire infrastructure build around the vehicle, excluding the competition test track, which is in the SpaceX headquarters in California. The entire pod is based on six major subsystems: Electrical, Software, Shell, Propulsion, and Brakes. Goal of Team Waterloop from the day of the team initiation is to design a complete and cost-efficient Hyperloop pod.

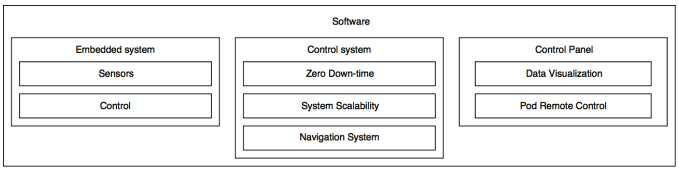


Fig. 1. Software system breakdown. Divide and conquer approach to break down software system into submodules that are later merged into a complete system

B. Problem statement

Working with a vehicle that is designed to operate at extreme speeds there is a variety of challenges that must be solved in order to provide complete control over the pod during the launch in the vacuum tube. Safety, being the main priority for any human method of transportation is subject to extremely high standards. The recipe to a successful pod launch during the competition, is built upon team's ability to precisely locate the pod inside the tube, and allow for a continuous control of the pod at all times during the launch. An additional challenge is introduced by the SpaceX competition rules, where participating teams are not allowed to alter the test tube to ensure the fairness of race. These are the challenges that team Waterloop was able to overcome in their system design.

C. Proposed solution

A divide and conquer approach was used to build the complex software system. The task was broken down into smaller problems: (1) Embedded, (2) Communication, (3) Control, that later were merged together into a single system. Fig. 1 on page 1 demonstrates the software system implementation breakdown.

Embedded system is directly responsible for collection and aggregation of data from all the on-board sensors and execution of control commands on various pod's control components.

Communication system is responsible for transmission of all the data between the control-panel and embedded system. Fig. 2 on page 2 provides a general overview of the system

ISSN 2305-7254

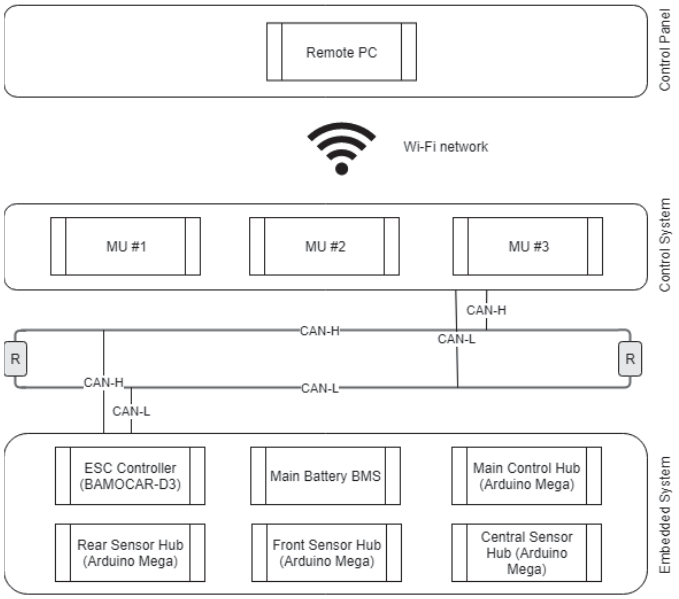


Fig. 2. System overview showing communication methods

that has been developed with all of the communication methods and main components displayed.

II. Detailed system overview

A. Embedded system

In abstract, embedded system is built to carry out two main tasks: (1) Collection of sensor data and (2) Execution of commands from control-panel. Since electronic components are spread out across many locations on the pod, use of a local network is required to allow for communication between all on-board sensors. The concept that is currently being used in cars with CAN-BUS networks will be used in the Hyperloop pod as well. Detailed overview of the entire pod's embedded-systems is given in Fig. 3 on page 3.

To develop a scalable system, a Master-Slave design will be used to create an abstraction of communication and data collection. In this case, two types of computing units will be used Master Units (MUs) (Master Units will be discussed in more details in Control system section) and Hub Units (HUs). With such design in place, the number of sensors on-board of the pod can be expanded easily by adding additional Hub Units to the main CAN-BUS network. Arduino Mega was the selected boards for Hub Units, due to it's low cost, off-the-shelf availability, expansion capability due to it's large community of enthusiasts around the world.

A CAN-BUS network will be built between Master Units and Hub Units, where Hub Units will have minimal logic and a single purpose of packaging low-level sensor output into binary packets and unpacking high-level control commands to low-level control components; and Master units will be used for data transmission to and from the control-panel.

1) Collection of sensor data with Hub Units: Embedded system will collect data from sensors through Hub Units. Fig. 5 on page 4 outlines the pod's sensor map. Hub Units are responsible for wrapping data into packets and transmitting the packets through CAN-BUS network. For efficient use of transmission channels, it is best to define a custom data structure. Table I on page 2 demonstrates a binary data structure that will be used to transmit data from Hub Units to Master Units. In order to develop a safe controlled pod, Waterloop has put together a list of sensors that must be installed on-board, which can be found in Table 4 on page 4.

TABLE I. Binary packet structure breakdown

[0:2]	[3:9]	[10:27]	[28:45]	[46:63]
3 bits	7 bits	18 bits	18 bits	18 bits
Packet Type	Packet Name	Data value 1	Data value 2	Data value 3

2) Execution of control-panel commands: In cases when a manual control is required over the pod, like initiating the pod launch script or engaging brakes in case of emergency conditions caused outside the tube. The control-panel will act as a client, responsible for sending commands to the Master Unit which will be the server in in communication channel. All control commands received by Master Unit will be forwarded to Hub Units (Control Hub specifically), through CAN-BUS which in place will be sent to pod control elements. Components such as Liquid Cooling, Friction-Drive Brakes, EC-Brakes, Rear-friction solenoids, Front-friction solenoids will be controlled through a series of relays that use Digital input to toggle elements' state On or Off.

B. Control system

Control system is responsible for three primary tasks: (1) Transmission of data to control-panel, (2) Execution of pod launch script, and (3) Execution of emergency braking procedures. The main computing device of control system is Raspberry Pi 3B used for all Master Units. The device was selected based on it's low cost, off-the-shelf availability, expansion capability due to it's large community of enthusiasts around the world and Linux operating system.

1) Transmission of data to control-panel with MUs: Master Units (MUs) will collect data from Hub Units, and use the collected information to adjust pod state during the launch. For efficient use of transmission channels, it is best to define a custom data structure. Listing 1 on page 2 demonstrates a JSON structure that will be used to transmit data from Master Units to control-Panel (Similar to binary data structure, but serialized into JSON object).

```
type CommPacketJson struct {
  Time int64      `json:"time"`
  Type string     `json:"type"`
  Name string     `json:"name"`
  Data []float32  `json:"data"`
}
```

Listing 1. Serialized JSON data structure written in Golang

To create a fast communication channel with minimal delay, multiple network protocols such as TCP, UDP and QUIC have been considered. While TCP loses to UDP in data transfer speed due to its need for a handshake process, UDP

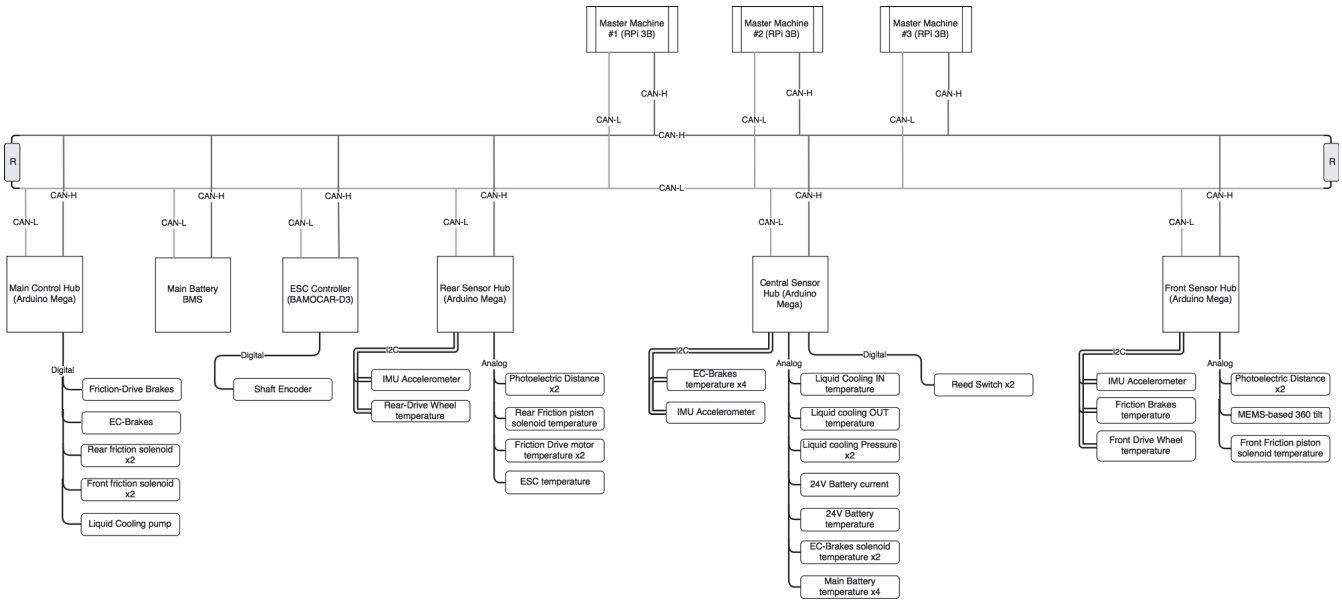


Fig. 3. CAN-BUS network as created in Waterloop's Goose 3 pod

also introduces a problem of unordered data streams in the communication channel [7]. Note, UDP has been recognized as the industry standard for all of the wirelessly controlled devices and vehicles due to its high data transfer speed, however the issue of unordered streams still persists. To solve this problem, Waterloop used QUIC protocol developed by Google and released to public in 2013 [8]. QUIC is built on top of UDP, but solves the problem of ordering data by introducing multiplexing of streams. Based on the results of series of tests described in Fig. 6 on page 4, the efficiency of multiplexing in QUIC protocol has been proved. Hence, a decision has been reached that each individual on-board sensor and control element will receive its own data stream.

2) Execution of launch script with Master Units: Master units will also be responsible for execution of pod launch script. Since, in its original design, the pod is developed to move at sonic speeds, it is unsafe to let a human pilot handle the task of controlling the pod. In addition, since the pod can only be communicated with through an in-tube wireless network, the entire pod control system has to account for the network interruptions that might occur in the tube. Hence, the safest solution is to create an entirely autonomous pod that will have an on-board machine to execute the pod launch script.

To create a script capable of controlling the pod at sonic speeds a powerful navigation system must be developed for the vehicle that will allow for precise pod positioning inside the tube. Section II-B4 on page 3 describes the navigation system in it's entirety. In addition, since Master Units are the computers responsible for running the pod launch script there is a need for redundancy measures that must be build to avoid pod stoppage in case of an unexpected Master Unit shutdown (Note, this is an important implication for a real-world scenario, because it would not be viable to stop all of the pods in the tube in case one of them experiences a Master Unit malfunction).

To solve this problem, Waterloop is using a Deterministic Finite Automata (DFA) that is distributed between multiple Master Units running in parallel. Fig. 7 on page 5 describes the pod state machine and triggers that cause state transitions. The consensus between Master Units network is achieved using a Raft Algorithm, [10]. The only change that is made to the algorithm is predefined initial node hierarchy instead of randomly generated one. Three Master Units on-board of the pod will be running the launch script with one of them as ACTIVE MU and others as IDLE. In case of the active MU FAILURE, heartbeat, established between all three MUs, will notify one of the other MUs to move into an ACTIVE state and continue execution of the launch script, while a watchdog on the FAILED MU will cause a MU reboot and set its state to IDLE once machine has restarted. Fig. 8 on page 6 describes the complete Master Unit recovery procedure. Therefore in case of a Master Unit failure, the system iterates through IDLE MUs without interruption of the launch script and only a simultaneous failure of all three Master Units will cause pod emergency braking procedures.

3) Execution of emergency braking procedures with Watcher Unit: In order to ensure the pod can be brought to a complete stop in case of failure of all Master Units (MUs), a Watcher Unit (WU) will be overseeing the state of all Master Units using heartbeat. In case the Watcher Unit doesn't detect heartbeat from any of the three Master Units, an emergency braking procedure will be triggered; where the main pod battery will be shut off and the physical failsafe brakes trigger will engage upon power shutdown. Fig. 8 on page 6 shows the conditions at which the Watcher Unit engages the emergency braking procedures. To ensure a complete independence of the Watcher unit from the rest of the system, the watcher unit will be powered separately using an independent Li-Po battery. Watcher Unit is built using a Raspberry Pi Zero, which has been chosen due to its smaller dimensions, low cost, and off-the-shelf availability.

- Sensor Hubs
  - Front Sensor Hub
    - IMU Accelerometer
    - Friction Brakes temperature
    - Front-Wheel temperature
    - Photoelectric Distance sensors x2
    - MEMS-based 360 Tilt sensor
    - Front friction piston solenoid temperature
  - Central Sensor Hub
    - IMU Accelerometer
    - Liquid Cooling pressure
    - 24V Battery current
    - EC-Brakes temperature x4
    - Liquid Cooling temperature IN
    - Liquid Cooling temperature OUT
    - 24V Battery temperature
    - EC-Brakes Solenoid temperature x2
    - Main Battery temperature x4
  - Rear Sensor Hub
    - IMU Accelerometer
    - Rear-Wheel temperature
    - Photoelectric Distance sensors x2
    - Friction-drive motor temperature x2
    - Rear friction piston solenoid temperature x2
    - ESC temperature
- Control Hub
  - Main control Hub
    - Liquid Cooling
    - Friction-Drive brakes
    - EC-Brakes
    - Rear friction solenoid x2
    - Front friction solenoid x2

Fig. 4: List of all sensors on-board of the pod distributed between sensor and control hubs. Note, different distributions are possible to create more redundancies, only one possible approach is shown here

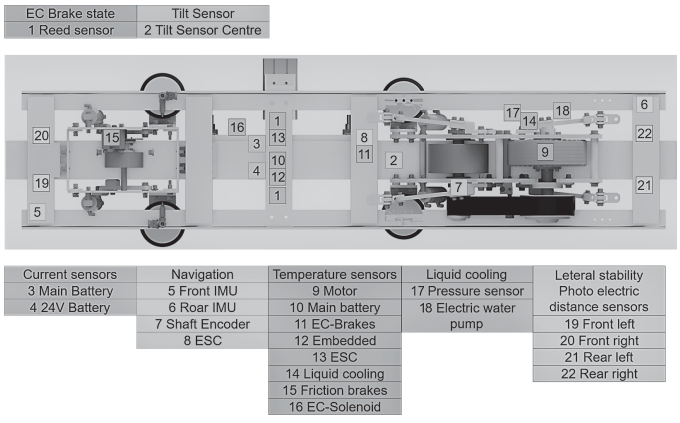
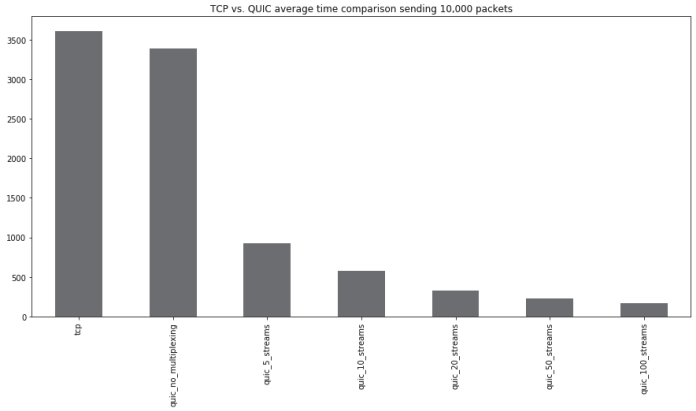
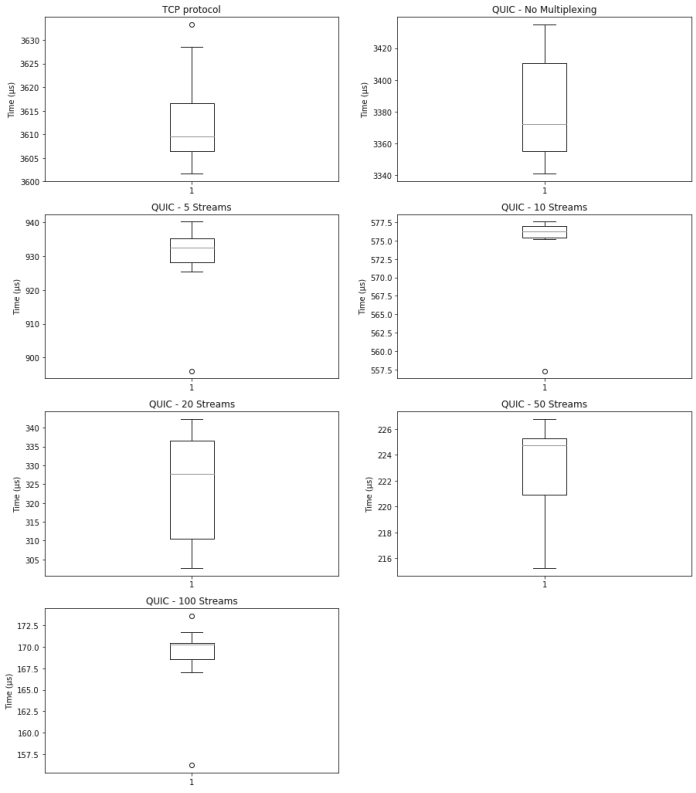


Fig. 5. Goose 3 sensor map

4) Navigation System: In order for Hyperloop to become a safe method of transportation it must have an accurate navigation system that will be used to determine the location of the pod in the tube. Fig. 9 on page 6 describes the data flow



(a) Histogram



(b) Box Plots

Fig. 6. TCP vs. QUIC with no multiplexing vs. QUIC with multiplexing

in navigation system. Note, the navigation system has proved to be a challenging task due to the restrictions of SpaceX competition, where no modifications can be applied to the test tube, which means that the entire navigation system must be completely self-contained in the pod.

To ensure accuracy of the system, an odd number of independent acceleration sensors will be used to create a basic majority voting system and a Kalman filter [9] will be applied after to remove spikes in data. Finally, for additional accuracy a shaft encoder will be installed on the main friction drive shaft to read wheel's rounds per minute (RPM).

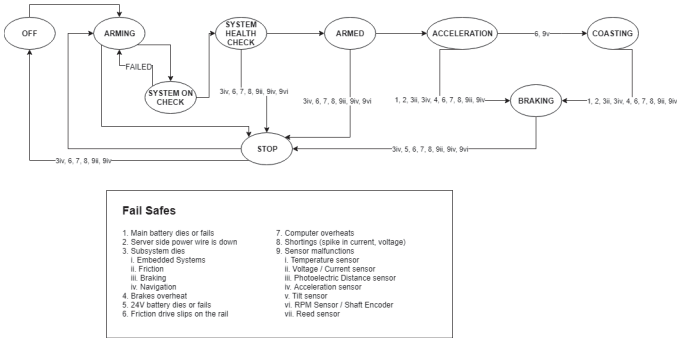


Fig. 7. Pod DFA state machine

Two other methods have been considered to locate the pod inside the tube. (1) Laser Range finder and (2) Color sensor and SpaceX in-tube color stripes. However, Laser Range finder can significantly lose its precision at long ranges as track imperfections constantly introduce changes to pod's pitch, roll and yaw, which can result in sensor reading wrong distance. As a solution to possible navigation needs, SpaceX provided in-tube color stripes that have a single purpose to allow teams have an in-tube navigation system. However, selecting a color sensor proved to be a challenge as the range of the sensor has to be adjusted with a third party lens, which is difficult to find; and all of the off-the-shelf sensors are designed for close range operation.

C. Control panel

In order to provide with control over the pod during the launch, a control-panel Graphical User Interface (GUI) will be developed. For best performance with QUIC packets, a desktop application will be developed for x64 Windows machines. The Control panel will provide with various pod control elements that will primarily be used for testing outside of the competition tube. While the entire pod launch script is executed from the on-board computers, having minimal control elements is vital for testing purposes. In addition, the Panel will be used for pod telemetry visualization purposes. Fig. 10 on page 6 shows the proposed Control Panel GUI.

D. Testing

Two types of testing procedures will be used to validate correctness of pod's software and embedded systems. Unit and Integration tests will be developed to test software, and Hardware in the Loop (HiL) tests will be used for testing of embedded systems. To run HiL tests, there is a need for a simulation environment that will allow running tests in an environment as close to real-life conditions as possible. Fig. 11 on page 6 describes a HiL testing rig that will be used to test pod's embedded systems. To control the simulation and verify test results, control-panel will be replaced with control-machine, sensors will be replaced with HiL transmitter that is capable of simulating individual sensors output (via I^2C connection) or send packets directly to CAN-BUS network and will take test input from a test computer.

To test individual sensors (that are not included in HiL testing) a series of Unit tests will be developed to test sensor accuracy and best installation methods.

III. Results

The system presented above addresses all of the challenges described in the initial problem statement. It allows for (1) Collection of sensor data, (2) Execution of control commands, and (3) Execution of pod launch script and provides with powerful testing features as well as post-launch analysis.

The single point of failure identified in the system is a failure of the CAN-BUS , which would result in a controlled breaking procedure initiated by Watcher Unit. The following procedure is the only safe method to keep the pod from damaging the tube and the team will use wire shielding to minimize the chance of CAN-BUS network getting damaged. Finally, every single element in the system can be tested through a series of unit tests or integration tests that will further decrease likelihood of failures.

IV. Future work

A. Switching CAN-BUS network to Ethernet network

The system can be further improved by switching CAN-BUS network on the pod for an Ethernet network. Ethernet (100 Mbps) has much faster bandwidth vs. CAN-BUS (1 Mbps) and can potentially create a much larger system or can potentially allow for faster sampling rate of sensors. However, since Ethernet networks are usually centralized through Ethernet Hubs, the change could potentially introduce new vulnerabilities to the system such as routers. All in all, the current pod, with existing number of on-board sensors, simply does not require such high-speed network and only introduces additional problems to the configuration and increases the costs (additional router and Arduino Ethernet shields costs)

B. Accommodating for multiple pods in the tube

Finally, the current system has been built with a single pod in mind (based on SpaceX competition guidelines). In the real-world application the system will be expanded to many pods in the tube at a time, which creates the need for inter-pod communication, as braking of one pod, will require the braking of all other pods traveling behind.

Hyperloop Alpha white-paper has theorized a minimum 6km distance between two pods at any time and a braking distance of 3km for each pod [11]. With these assumptions, the simplest solution would be to allow inter-pod communication so that the malfunctioning pod can notify others about its braking state. However the following system is reliant of pod's on-board computers that are constantly functioning in extreme conditions (experiencing acceleration or deceleration).

A better solution to the problem would be making changes to the Hyperloop tube which is a static object and that is capable of using much more reliable communication methods such as wired networks. In our vision, in an application outside of SpaceX competition, the tube will become subject to changes (currently SpaceX competition rules do not permit teams altering test tube), which will allow for augmentation of on-board pod navigation system with tube sensors that can track position of the pod with better precision and do not have to rely on the wireless network for communication. A number of sensors will be installed throughout the tube that will be able to track the position of all pods in the tube at the same

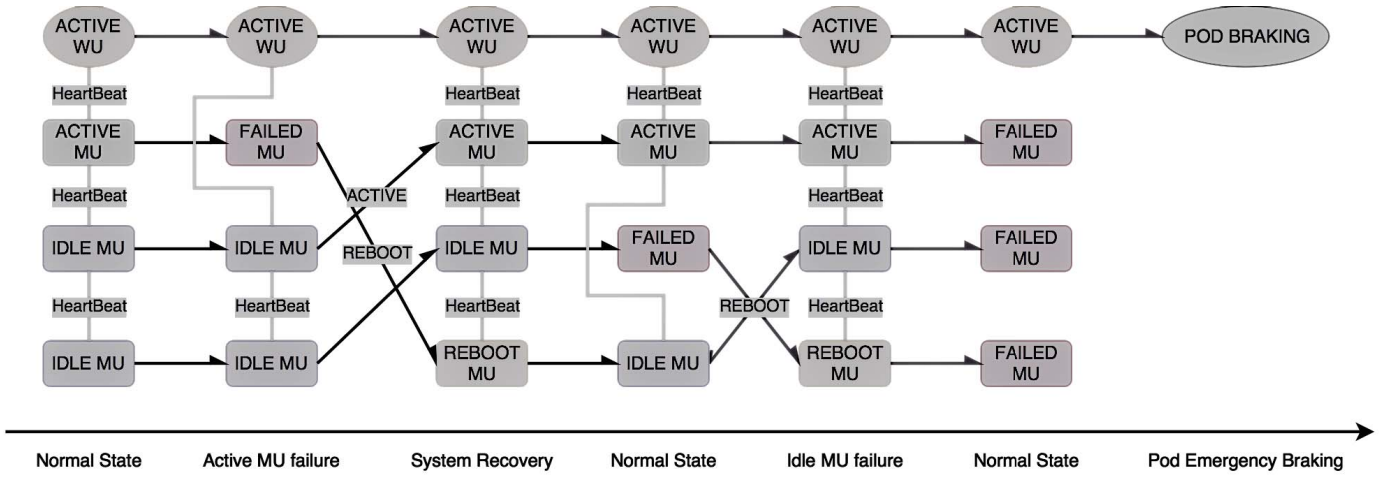


Fig. 8. MU recovery cycle. Complete transition cycle that allows for stateless recovery procedure

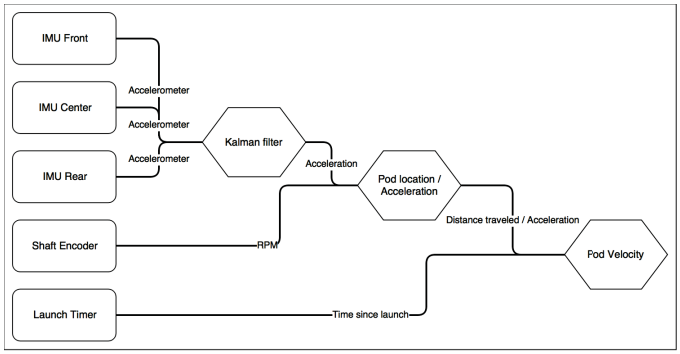


Fig. 9. Navigation system outline. Use of three IMUs is required to have a voting ability to verify the acceleration output of the sensors

Fig. 10. UI with minimal required control elements and real-time data visualization components

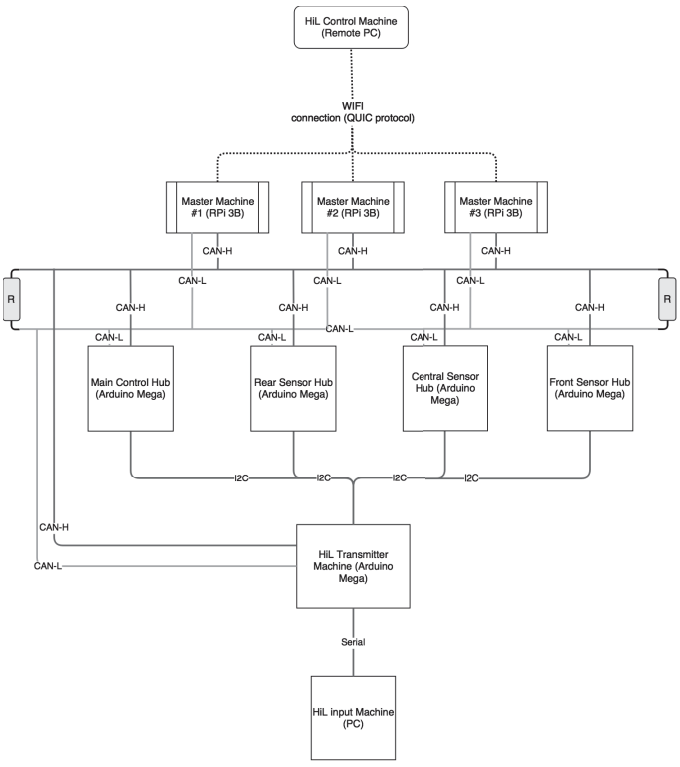


Fig. 11. HiL test rig developed to simulate real pod launch conditions and act as a complete system integration test

time (possible candidates for sensors are laser sensors due to their vacuum compatibility).

In this case, the system will use a similar approach to dispatcher centers currently used for airplane navigation. Fig. 12 on page 8 displays the communication system of the dispatcher and the pods as well as in-tube navigation sensors. Using a number of sensors installed throughout the tube: combination of time it took a pod to travel between sensors and the distance between the sensors will produce accurate pod velocity. The tube will provide the dispatcher with location of every pod in the tube as well as their speed. In case of one pod braking, the tube will detect braking pod's deceleration, will notify the dispatcher, which in turn will signal other pods traveling behind the malfunctioning pod to break.

Implementation of such solution will require creation of an Ethernet network between tube sensors and the dispatcher panel. Note in this case, the dispatcher panel is an extended

(added functionality to handle simultaneous control of multiple pods) version of control-panel previously described in the paper. The following solution has not been implemented by Waterloop yet, but has been thoroughly researched.

V. Conclusion

This paper presented the software and embedded design of the Goose pod, which participated in the SpaceX Hyperloop Competition from 2015 to 2017. As well as, a complete testing configuration and a look into possible future improvements that can be made. The system allows for very few points of failure and is capable of scaling to much larger pods as well as general transferability to other applications that require remote control over partially-autonomous vehicles. We can see the following system being used for control of various devices such as robots, rockets, trains, ships and more. The entire embedded system has been developed using off-the-shelf components such as Arduino Mega, Raspberry Pi 3B and Raspberry Pi Zero. All of the software is published under MIT license on Waterloop's Github account [5]. The total cost of the system has been kept under $300 (not including the costs of other embedded and electrical components such as sensors, batteries and control elements).

Acknowledgment

This projects successful outcome was the result of a massive team effort of the Waterloop, consisting team members [3] across Mechanical Engineering, Electrical Engineering, Computer Science, Business Management and Administration departments of Waterloo. This work could not have been completed without their contribution to the project.

The Waterloop was fortunate to have sponsors, supporters, press and Kickstarter campaign [6], who generously provided help and funding for this project, their support is hereby acknowledged and well appreciated. Additionally, we would like to thank SpaceX for organizing and hosting series of competitions.

References

[1] Goose - is a Hyperloop pod made by Waterloo University team, *Goose X*, (Web-page). https://teamwaterloop.ca/racer-goose-ii/

[2] Official web-page of University of Waterloo https://uwaterloo.ca/

[3] Waterloop - team members, *Waterloop Team,*, (Web-page). https://teamwaterloop.ca/team/

[4] SpaceX, *SpaceX Hyperloop Test-Track Specification,*, SpaceX.(Online Article), 2016. https://cdn.atraining.ru/docs/TubeSpecs.pdf

[5] Official page of Hyperloop Goose project with open-source materials https://github.com/waterloop

[6] Kickstarter, *Waterloop: The Canadian SpaceX Hyperloop Competition Team,*, (Web-page). https://www.kickstarter.com/projects/1629380361/waterloop-the-canadian-spacex-hyperloop-competitio

[7] *Comparing TCP and UDP Speed and Packet Loss Over LAN and WAN* http://milliways.bcit.ca/res/report.pdf

[8] *QUIC, a multiplexed stream transport over UDP* https://www.chromium.org/quic

[9] *A New Approach to Linear Filtering and Prediction Problems* 1960 https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[10] In Search of an Understandable Consensus Algorithm, https://raft.github.io/raft.pdf

[11] Hyperloop Alpha white paper http://www.spacex.com/sites/spacex/files/hyperloop_alpha.pdf

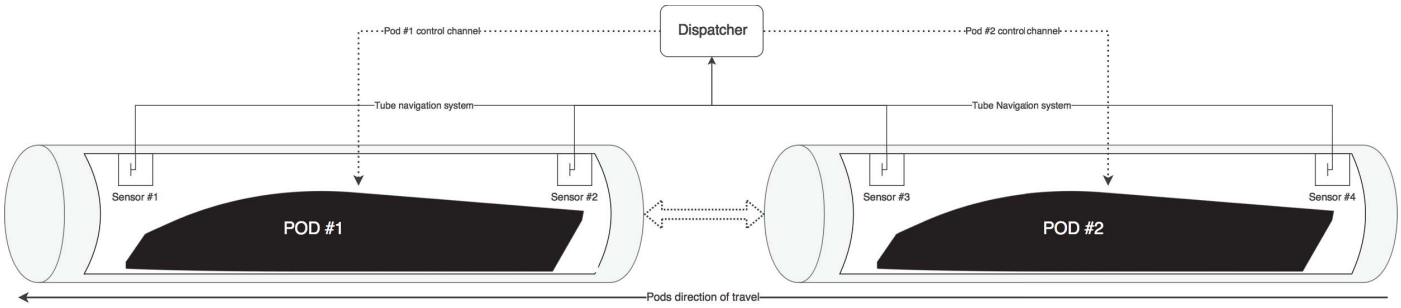


Fig. 12. Pod-Tube-Dispatcher communication system