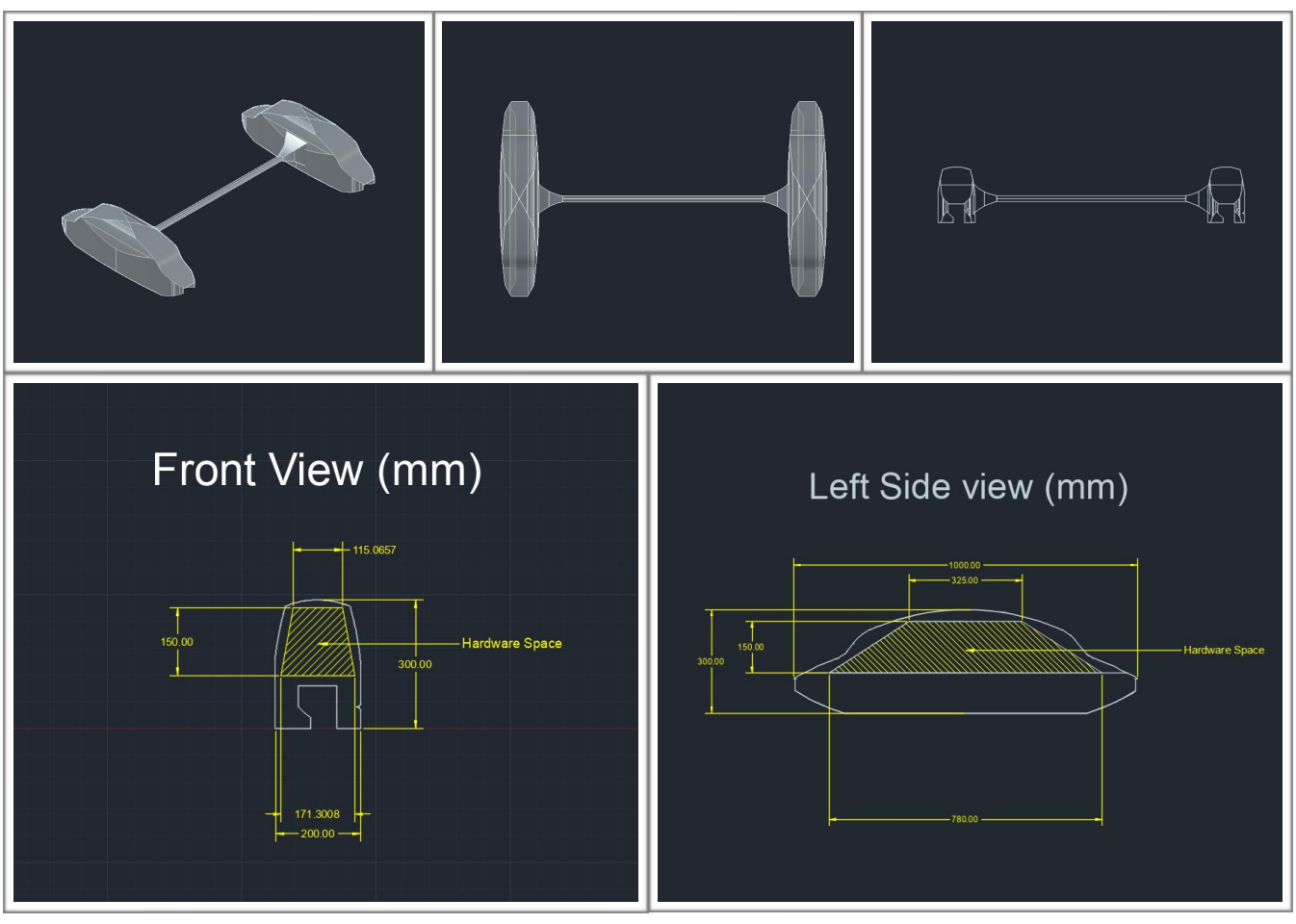
Hardware Design

Project Outline for Aaron



Hello Aaron, our team successfully was able to get all four people for the completion of this project, which is awesome! That being said, there is a lot of room for confusion, misinterpretation, and error, since all four of us are trying to tackle one idea! Therefore, for organizational purposes, I have attached a table below, outlining what you should complete in order for this hack to go smoothly.

Please read the table below beforehand, so that when the hackathon begins, we can hit the ground running!

(Please note, I am not trying to micromanage the team and you are free to make changes! However I am doing this so there is consistency between the hardware and the software components of this project. Have fun!)

	Tasks	Description

Part A	Create a fully functional circuit schematic, as well as a VHDL/ Verilog file that demonstrates that it works.
	
	Please see the notes for what the device hardware must do.
	
	Once the files are completed, upload them to our repository.
	
	The micro-controller we will be using is Raspberry Pi
		• <u>The device must</u>:
		  • Be able to travel at least 200km on one charge, going an avg. of 100km/h. (Acceleration time is not important)
		  • Detect objects in between the rails (Using the central beam. Avoid ultrasonic sensors)
		  • Detect any objects on the rail within 50m in front of the device
		  • Use two LQ infrared sensors to sense heat coming from the left and right sides of the device (Used to detect animals/people near the tracks) NOTE: Make sure that the sensor does not pick up the heat of the RODD (Railway Obstacle Detection Device) itself.
		  • Be equipped with a camera, on both devices, facing both rails in order to detect metal fatigue. An ultraviolet light must shine near the camera in order for the fractures to be visible
		  • Detect if the device is moving or not. If not, there might be an obstacle in the device's way preventing it from moving and posing a hazard to the train.
		  • Be equipped with a transceiver that can communicate with other transceivers within 5km.
		  • Be equipped with a bright flashing white light to warn others of the device's presence, as well as a bright flashing red light to indicate an emergency. (Consider using xenon lights, similar to those on top of school buses.)
		  • Be able to turn on/off via the micro-controller. The Raspberry Pi itself does not turn off, just the other components of the RODD. A physical switch must be used to turn the Pi off.
		• All signals from the sensors will be inputed by the micro controller. Simply indicate: In_1, In_2 … In_n on the schematic. These will represent input pins of the micro-controller.
		• The following are the components that will be controlled by the micro-controller:
		  • The motors moving the RODD
		  • The transceiver (Both receiving and transmitting signals to & from the micro-controller)
		• To indicate outputted signals that will be controlling the components listed above, simply indicate: Out_1, Out_2 … Out_n on the schematic.

Part B	Using Java, create a library with methods that interact directly with the hardware of the device.
	
	This library will be used by Teresa, our team's software developer. That is why it is important that the main methods are established, so that you and Teresa can work on programming simultaneously.
	
	For example, one method would be *public static int move(int speed),* where it would move the device's motors and return the current speed of the RODD.
		• <u>Please include the following functions/methods:</u>
		  • **int move(int speed):** Accepts the speed in km/h as an argument (negative if reversing), moves the RODD, then returns the current speed of the device as a signed integer. Return infinity or negative infinity if the device does not accelerate (meaning it's stuck) for more than 10 seconds.
		  • **bool object_between_tracks():** Returns true or false wether or not an object between the tracks is detected. This function, along with most other sensor functions, will be running on an interval of ~200ms.
		  • **bool obstacle_on_rails():** Returns true or false other or not an obstacle within 50m of the device is detected.
		  • **read_infrared_cameras():** Using a library that communicates with the infrared, return the raw data of the camera reading. Return the data in an array, with each reading having it's own key, "left" and "right". (Please correspond with Teresa to determine which data type will be used)
		  • **read_rail_cameras():** Turns on the ultraviolet light facing the rails, then reads the camera value (again, just like the infrared camera, use a library of your choice. Also, return an array of two elements with keys "left" and "right". Tell Teresa the data type used for the return value).
		  • **string transceiver_in():** loads serial data from the buffer on the onboard transceiver. Use a library of your choice and return the value as a string.
		  • **bool transceiver_out(string data):** Send data out of the onboard transceiver. Return true or false wether or not the data was successfully sent.
		  • **bool power(int mode = 2):** Turns the RODD on or off, depending on the "mode" parameter. 0 = off, 1 = on, 2 = toggle. Returns the new state of the device (false for off, true for on).
		  • **bool light(int light, int mode):** Turns a light on/off and returns the new state. "light" parameter selects the light to turn on/off (0 = flashing white light, 1 = UV light for rail camera, 2 = red emergency indicator light). The "mode" parameter determines what to do with the light (0 = off, 1 = on, 2 = toggle)
		• For error handling, please call "bool rodd_error(string error_message)". Teresa will write this function.