- TCP communication between the car and the dashboard is done through socket programming.
- Once a client node starts, it requests a connection to the python server using the address and port of the server.
- Once a connection is established, the client sends it's type to the server (string). Upon receiving the type, the server sends an acknowledgement (string) to the client. After receiving the acknowledgement, the client will be able to send messages to the server.
- Currently the server accepts 2 clients which are embedded in the camera and controller ros node
- The protocol is capable of end to end communication for some of the datatypes
- The client and server classes have public methods exposed for sending datatypes (refer to doxygen docs)
- The client and server also handles receiving and decoding message in the background. The messages are stored in queues for each datatype. These queues can be either accessed directly in python or through getters in c++.
  (refer to doxygen docs)

- Datatypes are encoded into bytes so that they can be transferred over the network.
- Here is an example of a packet for a simple datatype.
- We simply form the packet by allocating 4 bytes for the messages size, 1 byte for the datatype and the rest is composed of the serialized datatype's bytes which is either obtained through standart library decomposition into bytes or through the ros serialization library.

Packet :	Size (4 bytes)	Datatype (1 byte)	Datatype bytes

- When the packet is received, we simply parse the back into datatypes/objects using standart library or ros serialization library
- The reason we send the length of the message first is so that the receiver knows how many bytes to read afterwards.

- Objects composed of multiple datatypes which does hot have a serialization protocol must be decomposed into individual serializable components.
- Here is an example of a packet for custom objects

Packet :	Size (4 bytes)	Datatype (1 byte)	Array of lenghts (n x 4 bytes)	Array of bytes for each datatypes (n x m bytes)

- Once the primitive datatypes are received, the object is then reconstructed and stored.
- Here is a table of the current supported datatypes.

0x01	string
0x02	sensor_msgs/Image (rgb)
0x03	sensor_msgs/Image (depth)
0x04	std_msgs/Float32MultiArray (road objects)
0x05	std_msgs/Float32MultiArray (waypoints)
0x06	std_msgs/Float32MultiArray (signs)
0x07	std_msgs/String (message)
0x08	Object (go_to_srv)
0x09	Object (go_co_cmd_srv)
0x0A	Object (set_states_srv)
0x0B	Object (waypoints_srv)

- The protocol can support up to 256 different datatypes
- For each new datatype you must implement both encoding and decoding protocol specific to that datatype on both the client and server, as they operate using different programming languages. e.g send_datatype(...) and parse_datatype(bytes)