Robot Programming

Sensors MARRtino

Giorgio Grisetti

Outline

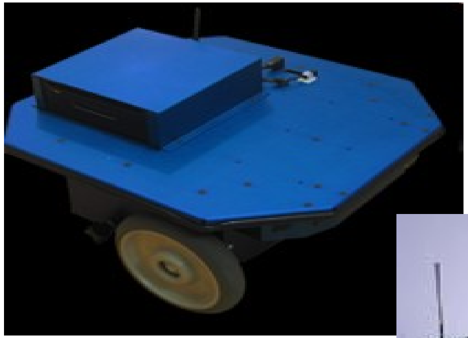
- Robot Devices
  - Overview of Typical sensors and Actuators
  - Operating Devices in ROS
- Mobile Bases
- MARRTino
  - Hardware
  - Firmware

Mobile Base

- A mobile platform is a device capable of moving in the environment and carrying a certain load (sensors and actuators)

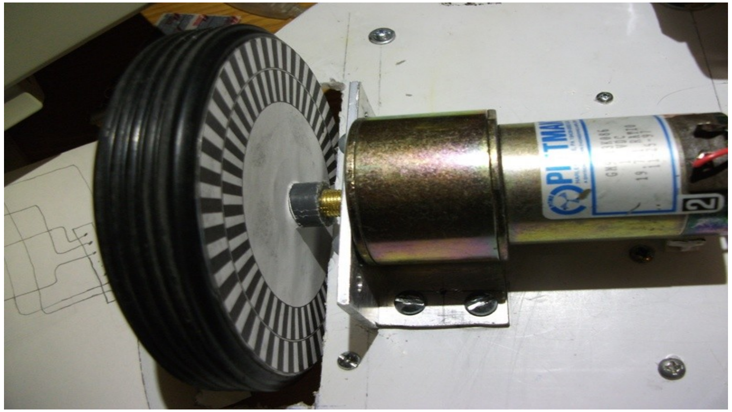
- At low level the inputs are the desired velocities of the joints, and the output is the state of the joints

- At high level it can be controlled with linear/angular velocity, and provides the relative position of the mobiel base w.r.t. an initial instant, obtained by integrating the joint’s states (odometry).



Proprioceptive Sensors for Ego-Motion

- Wheel encoders mounted on the wheels
- IMU:
  - Accelerometers
  - Gyros



- The estimate of ego-motion is obtained by ***integrating*** the sensor measurements of these devices. This results in an accumulated drift due to the noise affecting the measurement
- In absence of an external reference there is ***no way*** to recover from these errors

Exteroceptive Sensors

Perception of the environment

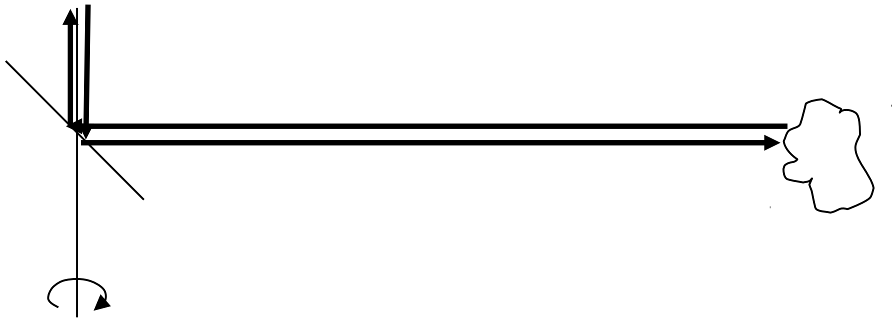
Active:
- Ultrasound
- Laser range finder
- Structured-light cameras
- Infrared

Passive:
- RGB Cameras
- Tactiles

Time of flight

Intensity-based

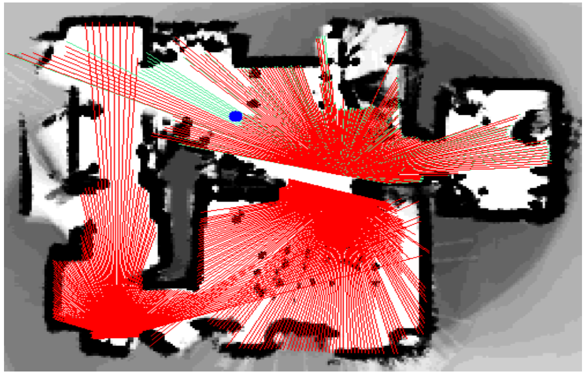
Laser Range Scanner

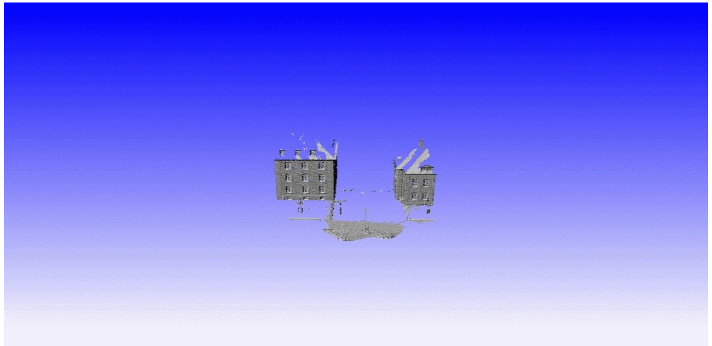
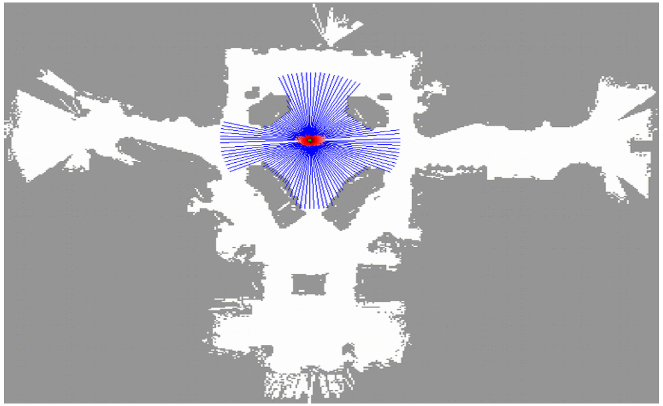
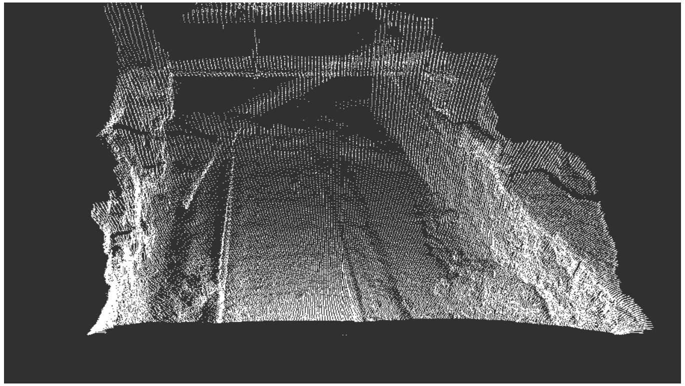
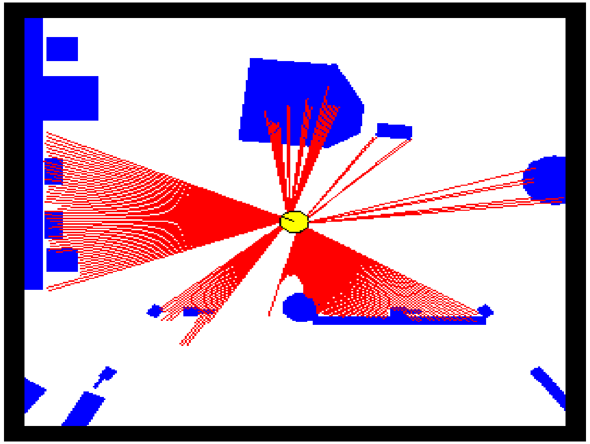


Properties

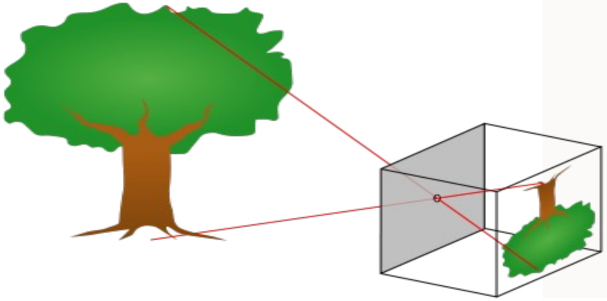
- High precision
- Wide field of view
- Approved security for collision detection

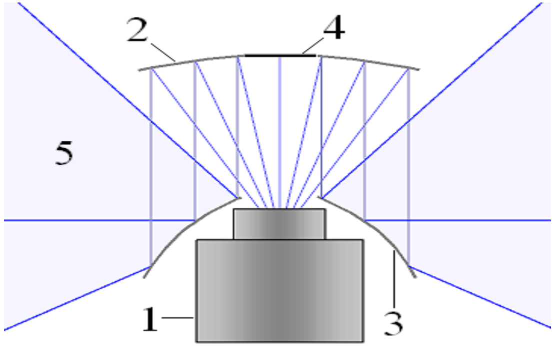
Typical Scans

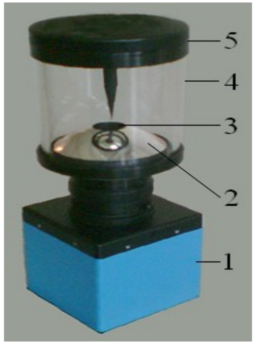


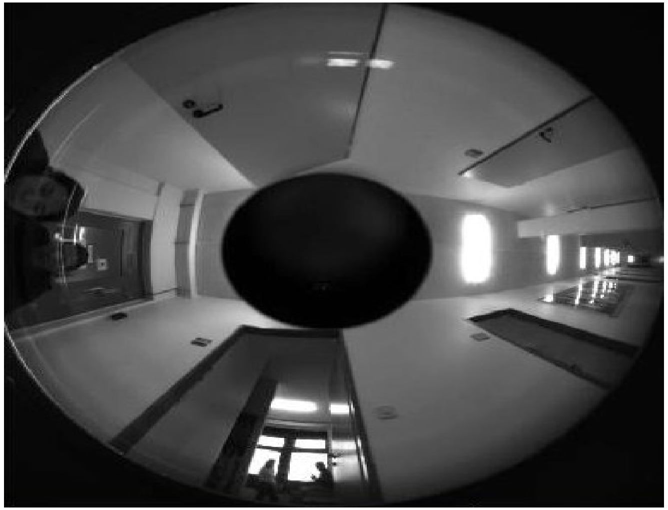





RGB Monocular Camera





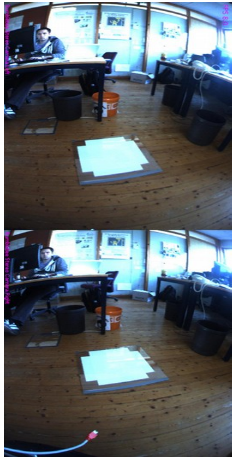


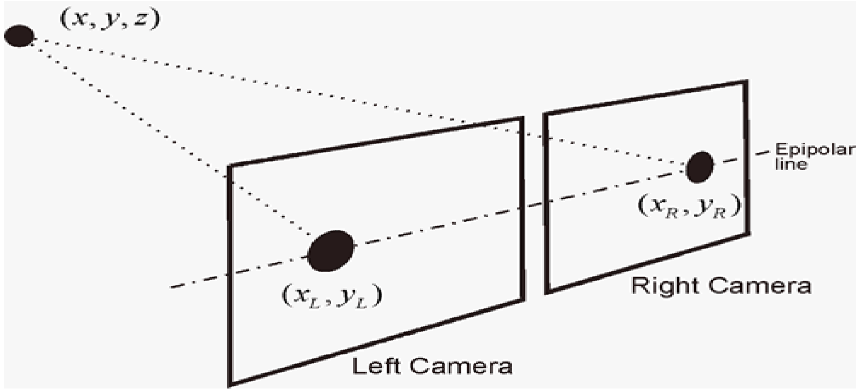


RGB Monocular Camera

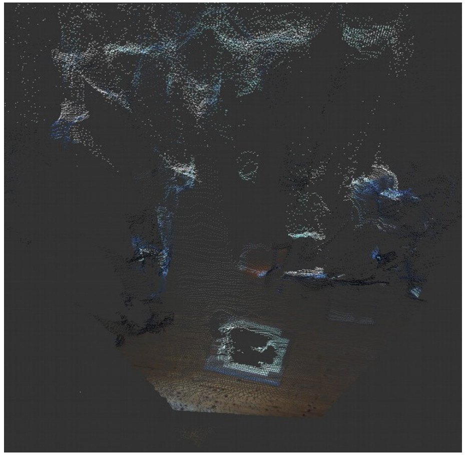
- Cameras measure the intensity of the light projected onto a (typically planar) ccd through a system of lenses and/or mirrors
- Provide a lot of information
- Project 3D onto 2D, which results in the unobservability of the depth
- The scene can eb reconstructed by multiple images (see SfM)

Stereo Camera





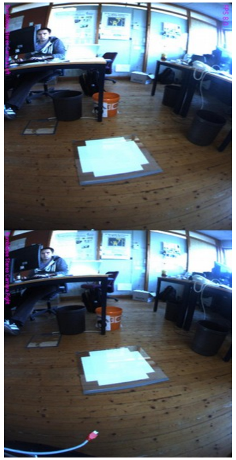
reconstruction from top

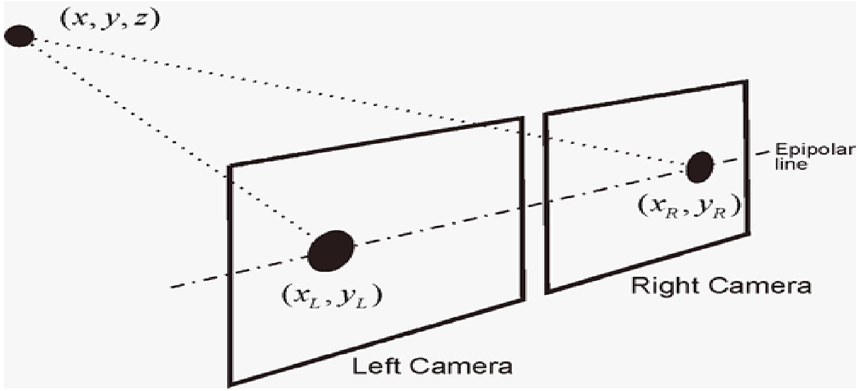


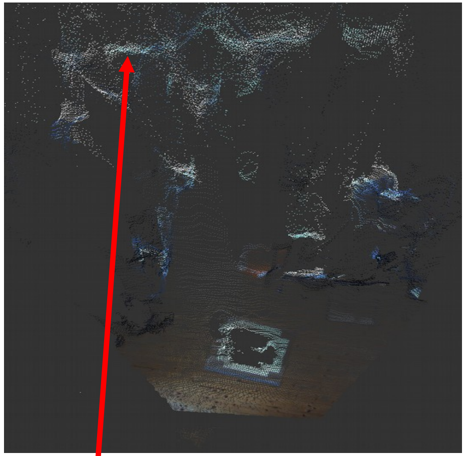
- Stereo cameras are combination of 2 monocular cameras that allow triangulation, given a known geometry.
- If the corresponding points in the images are known, we can reconstruct the 3D scene.
- Error in the depth depends on the distance!
- Sensible to lack of texture

Stereo Camera

reconstruction from top



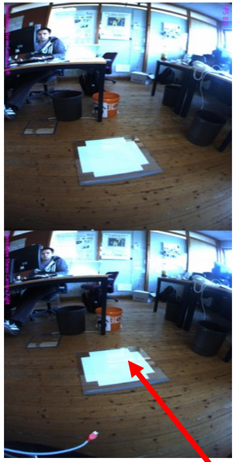


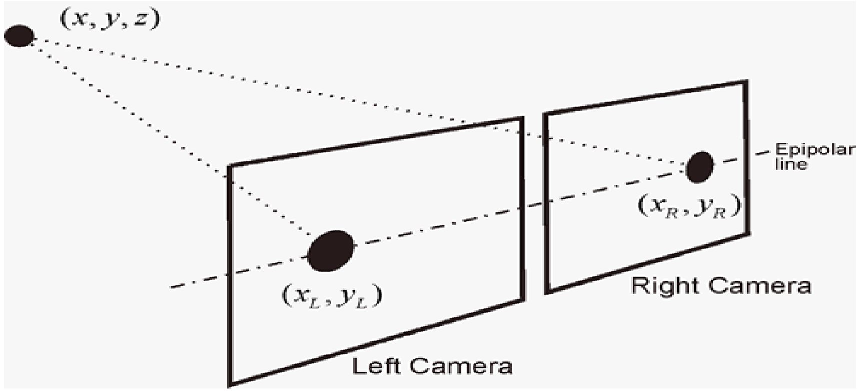


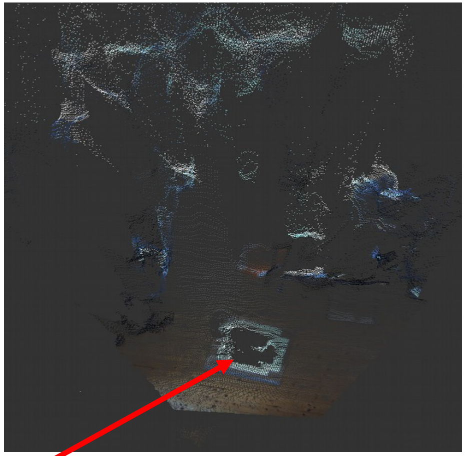
- Stereo cameras are combination of 2 monocular cameras that allow triangulation, given a known geometry.
- If the corresponding points in the images are known, we can reconstruct the 3D scene.
- Error in the depth depends on the distance!
- Sensible to lack of texture

Stereo Camera

reconstruction from top







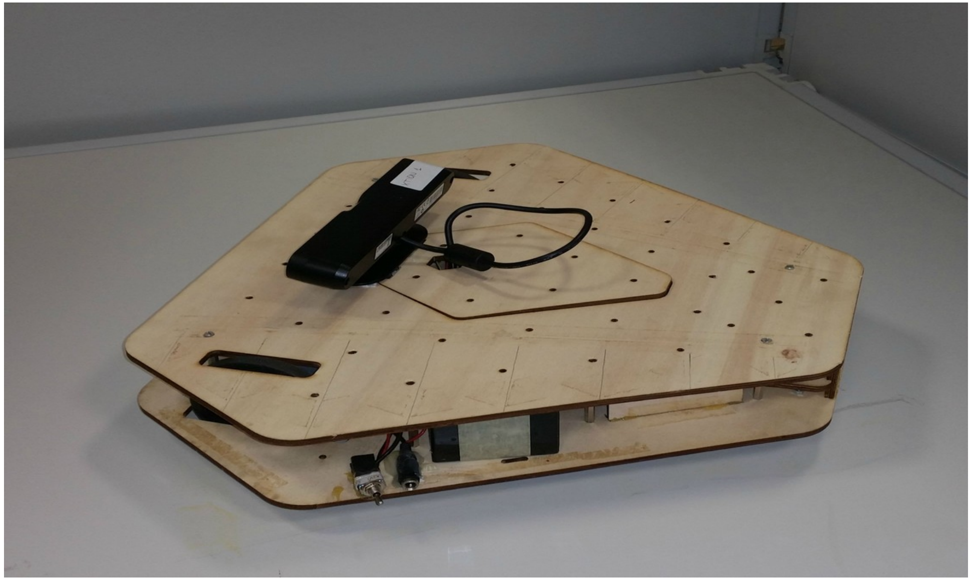
- Stereo cameras are combination of 2 monocular cameras that allow triangulation, given a known geometry.
- If the corresponding points in the images are known, we can reconstruct the 3D scene.
- Error in the depth depends on the distance!
- Sensible to lack of texture

MARRtino

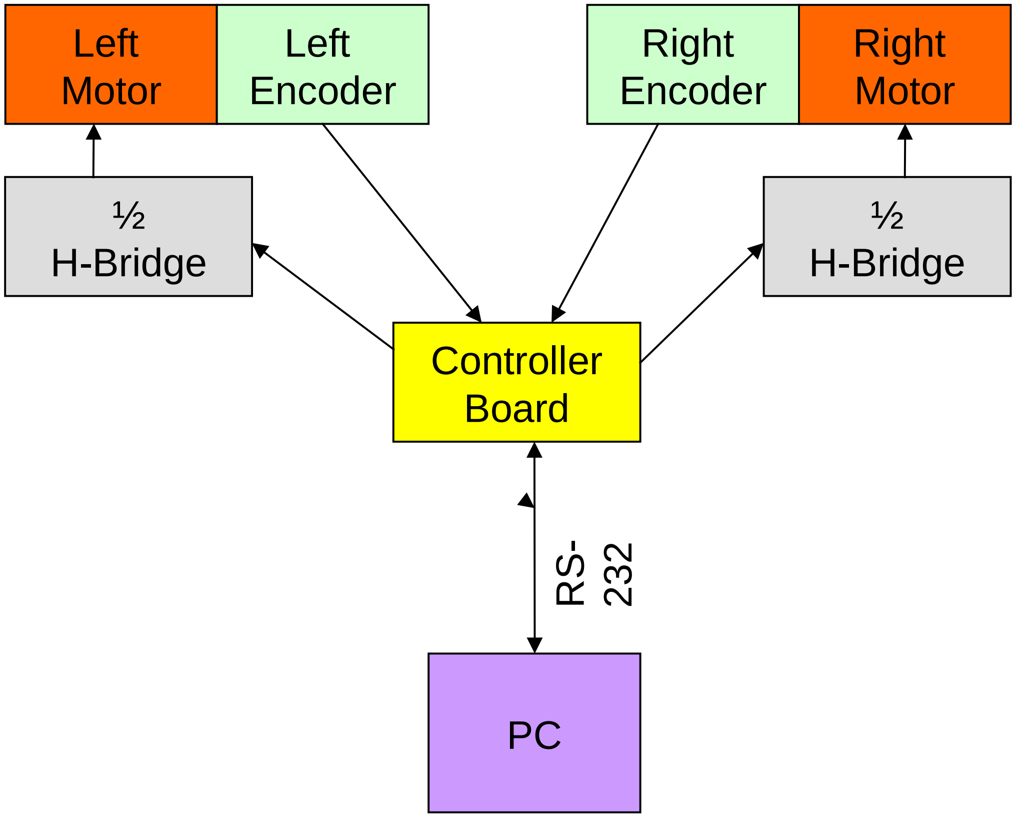
- Is a simple but complete mobile base designed to be used in the MARR course.
- The cost of the parts is around 300 euro
- It is entirely open source
- It is integrated in ros through a simple node that publishes/subscribes standard topics

MARRtino

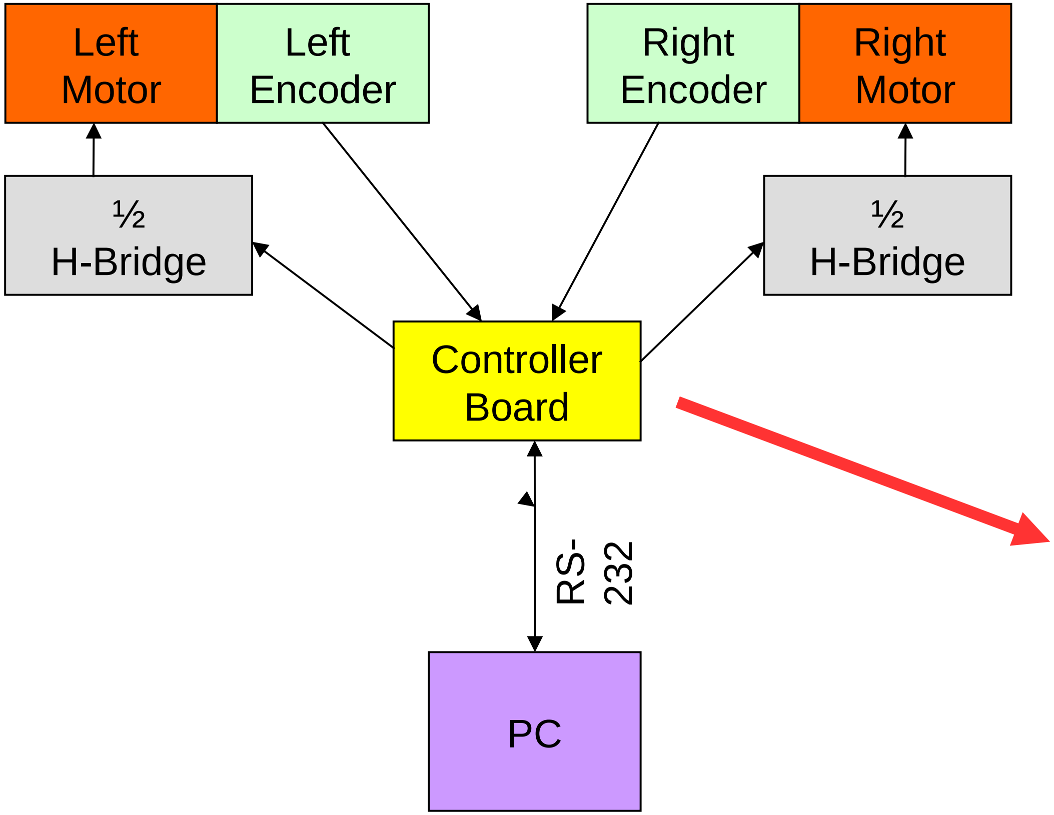
- Is a simple but complete mobile base designed to be used in the MARR course.
- The cost of the parts is around 300 euro
- It is entirely open source
- It is integrated in ros through a simple node that publishes/subscribes standard topics

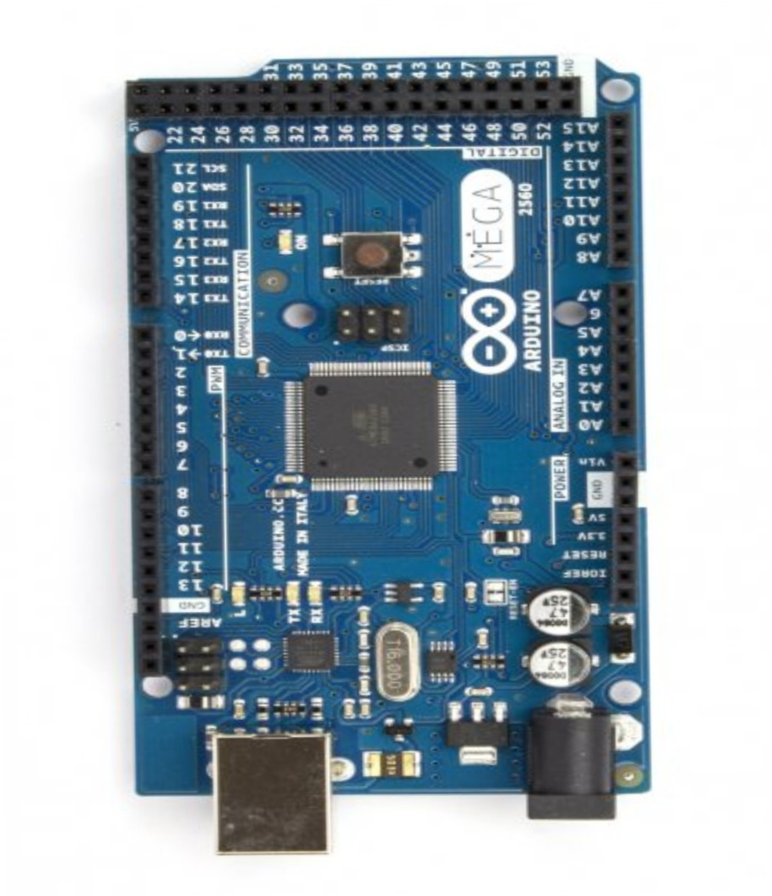


MARRtino: Electronics



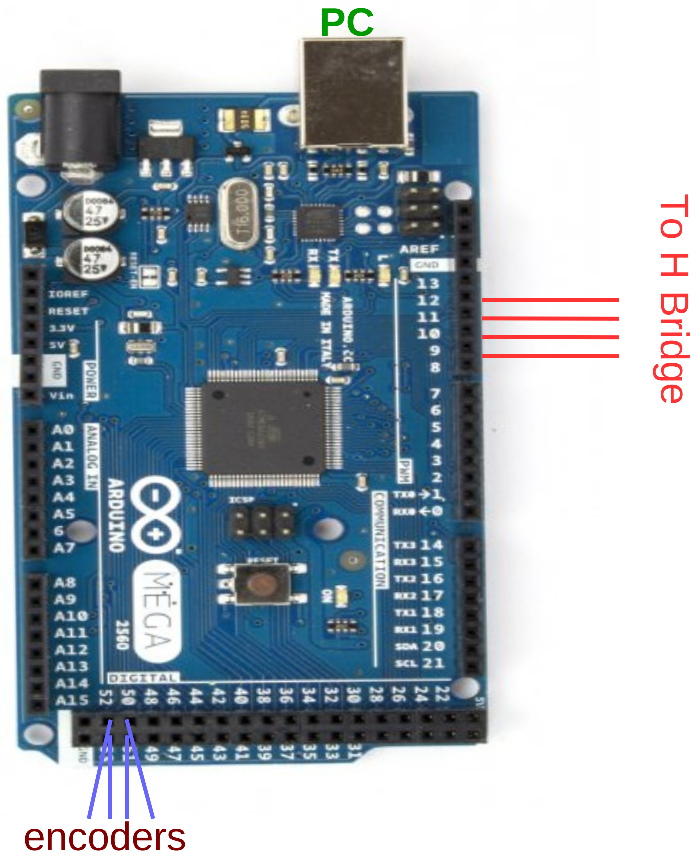
MARRtino: Electronics





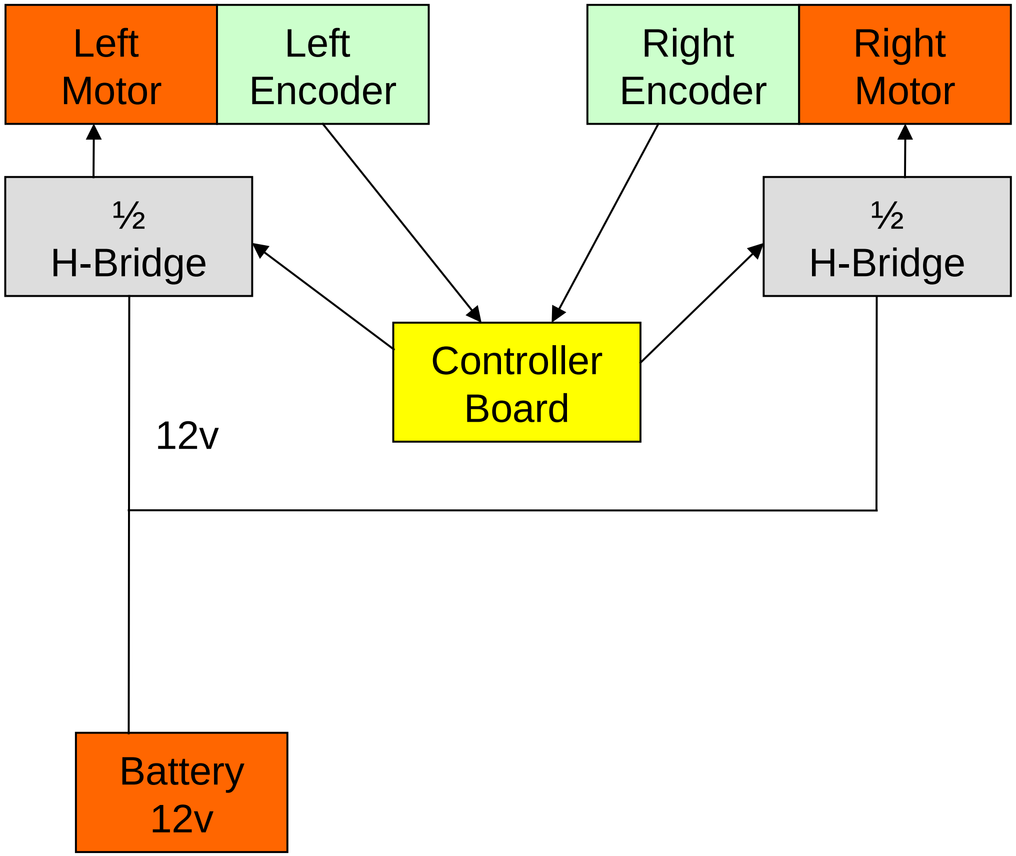
MARRtino: Electronics

- The PC communicates with arduino through USB
- Each encoder provides two signals
- Each PWM requires at least 2 wires
- The wiring of the PWM depends on the H-Bridge used



MARRtino: Power

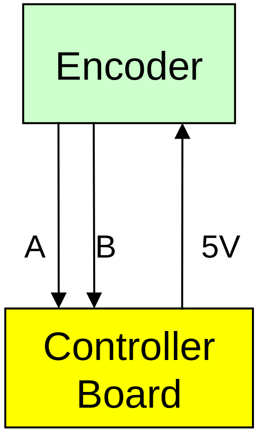
- H bridges: 12 V from both batteries, 5V from logic generated from internal regulator
- The system can either charge the batteries or be powered ON.



Controller is powered through USB
Controller and H bridges share the GND

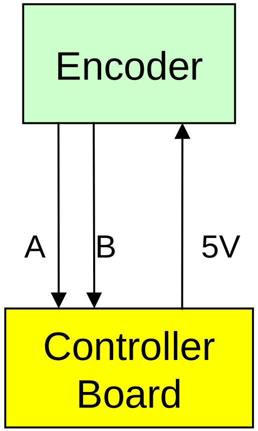
MARRtino: Encoders

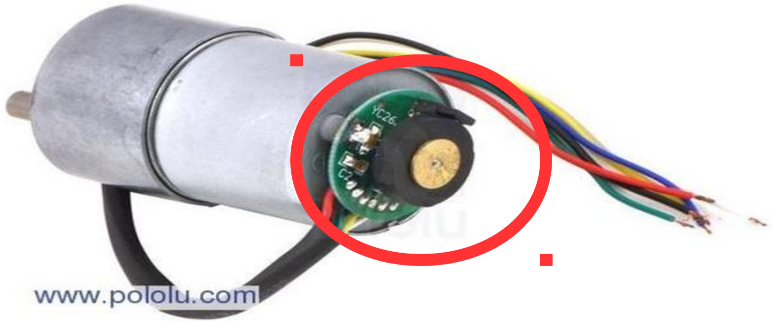
- Each encoder has two signals (A, B) and requires a 5V voltage supplied by the controller board
- The encoders are managed by the Quadrature Encoder Module (QEI) of the controller, that takes care of counting ticks and direction



MARRtino: Encoders

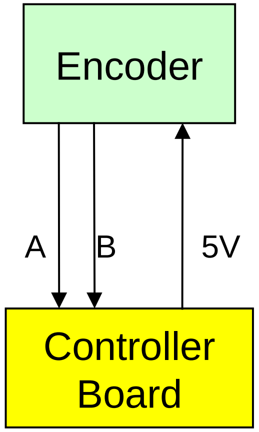
- Each encoder has two signals (A, B) and requires a 5V voltage supplied by the controller board
- The encoders are managed by the Quadrature Encoder Module (QEI) of the controller, that takes care of counting ticks and direction

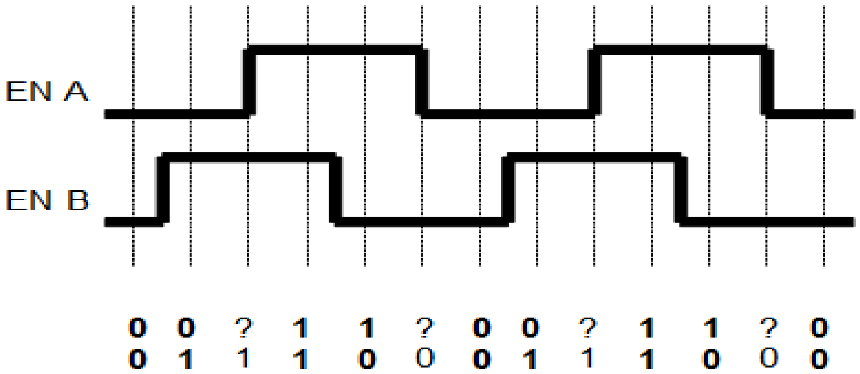




MARRtino: Encoders

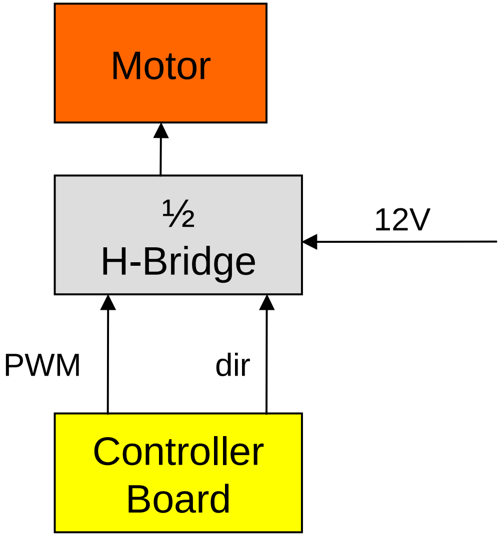
- Each encoder has two signals (A, B) and requires a 5V voltage supplied by the controller board
- The encoders are managed by interrupt on edges, that takes care of counting ticks and direction

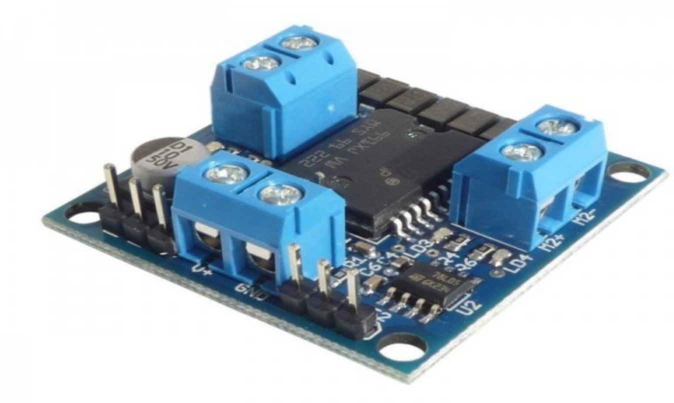




MARRtino: H Bridge

- The motor is connected to the H Bridge, that provides the necessary voltage and current to drive it.
- The H bridge requires 12V power directly from the battery
- The controller board controls the H bridge by*
  - A square wave whose duty cycle is proportional to the voltage applied to the motor, that controls the speed (PWM)
  - A direction pin, that reverts the voltage when asserted, causing the motor to rotate in the opposite direction

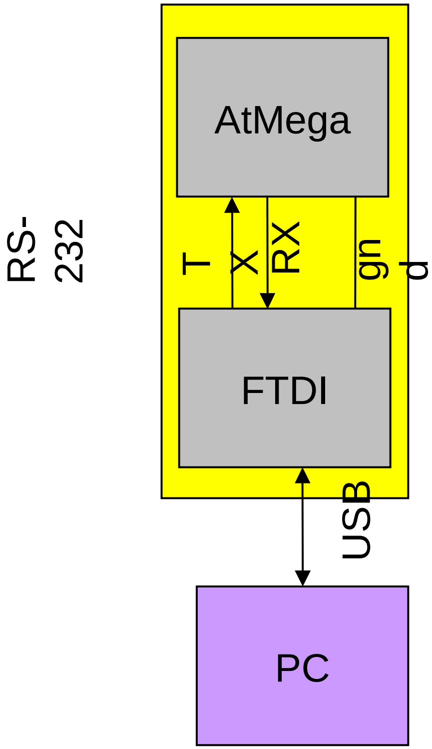




* Other modes are possible, please refer to the wiki for details

MARRtino: PC connection

- The robot communicates with the PC through an RS232 interface at TTL levels (0-5V)
- The TTL-RS232 is converted in USB through an FTDI chip
- The device is visible on Linux as /dev/ttyXXX



MARRTino: Firmware

- The controller runs an event driven C program that
  - Executes PID controllers on both wheels
  - Integrates the odometry
  - Communicates with the PC
  - Implements a watchdog
- The PC periodically sends to the controller the desired translational/rotational velocities
- The controller periodically sends to the PC state packets that include the integrated odometry since the last message and the battery state
- If the controller does not receive any message from the PC for a while it sets the speed to 0 (safety measure)

MARRTino: Wiki

- The wiki describes all the steps to:
  - obtain the firmware code
  - flash the firmware on the board
  - run the web client and connect the parts while configuring the board
  - Starting a ROS node

  - **https://www.marrtino.org/**
  - **https://gitlab.com/srrg-software/srrg2_orazio**

More about Firmware

- Orazio: is the firmware for a mobile robot developed mainly by Giorgio, running on the control board
- Features
  - multi architecture (ATMega, dsPIC)
  - supports a variable number of joints
  - support sonars
  - layered architecture
    - current/joints/platform

Orazio: Subsystems

Modules running on the control board

- System: communication, watchdog, packet selection
- Joints: control of the motors at joint level
- Drive: control of the platform (base velocities/odometry)
- Sonar: sonars
- Servo: up to 3 servos
- IMU: Interface to an on board IMU

Each subsystem has

- a status (updated by the program on the control board and sent periodically to the host)
- parameters (stored in EEPROM)
- can receive commands
- can be enabled or not
- can send updates or not

Packets

```
// here are the limits
typedef uint8_t PacketType;
typedef uint8_t PacketSize;
typedef uint16_t PacketSeq;

typedef struct {
  PacketType type;
  PacketSize size;
  PacketSeq  seq;
} PacketHeader;

typedef struct {
  PacketType type;
  PacketSize size;
  PacketSeq  seq;
  uint8_t system_enabled;
  uint8_t update_enabled;
  uint8_t _padding[2];
} ParamPacketHeader;

#define PARAM_CONTROL_PACKET_ID 1
```

```
....
//! sent from the pc to the robot causes
//! the robot to move at tv/rv
#define DIFFERENTIAL_DRIVE_STATUS_PACKET_ID 15
typedef struct {
  PacketHeader header;
  float translational_velocity;
  float rotational_velocity;
} DifferentialDriveControlPacket;

//! sent from the pc to the robot causes
//! the robot to set these parameters
//! from the robot to the PC as a request from
#define DIFFERENTIAL_DRIVE_PARAM_PACKET_ID 15
typedef struct {
    ParamPacketHeader header;
    float ikr;
    float ikl;
    float baseline;
    float max_translational_velocity;
    float max_translational_acceleration;
    float max_translational_brake;
    float max_rotational_velocity;
    float max_rotational_acceleration;
    uint8_t right_joint_index;
```

Orazio: Packets

- The communication between host and controller happens by sending serial packets (see orazio_packets.h)
- Each packet has a unique id, and carries one type of information

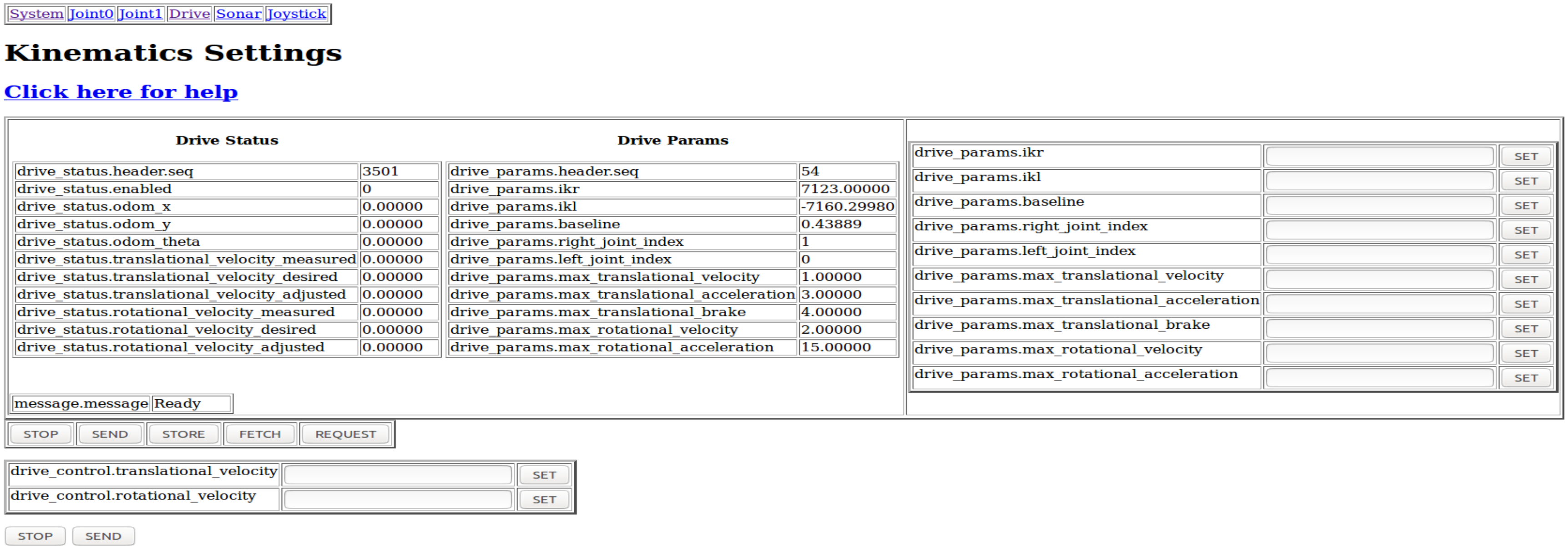
- Packets from PC:
  - issue commands (e.g. setSpeed)
  - control parameters
    - set (write)
    - save (to EEPROM)
    - restore (from EEPROM)
    - get (read)
- Packets from robot
- Status of subsystems
  - Parameters
  - Responses

Host Interaction

- Orazio sends periodically at a configurable rate status packets of all subsystems of interest
- The host reads these packets, and the "client" program stores the last packet of each type in a variable (for later inspection)
- All packets of a same interval constitute an "epoch"
- If needed, the client sends some command to the host
- After each epoch, the client sends a "keepalive" packet to notify the control board it is still running
- OrazioClient is the "C" class that implements these functionalities

WEB interface

- Client application that communicates with orazio
  - used to set the parameters, and issue controls
  - program starts a local web server
  - from host_build
    - orazio -serial-device /dev/ttyACM0 -resource-path ../html
    - Hint: flash the firmware and follow the instructions to connect the devices



System | Joint0 | Joint1 | Drive | Sonar | Joystick

Kinematics Settings

Click here for help

Drive Status	
drive_status.header.seq	3501
drive_status.enabled	0
drive_status.odom_x	0.00000
drive_status.odom_y	0.00000
drive_status.odom_theta	0.00000
drive_status.translational_velocity_measured	0.00000
drive_status.translational_velocity_desired	0.00000
drive_status.translational_velocity_adjusted	0.00000
drive_status.rotational_velocity_measured	0.00000
drive_status.rotational_velocity_desired	0.00000
drive_status.rotational_velocity_adjusted	0.00000

Drive Params	
drive_params.header.seq	54
drive_params.ikr	7123.00000
drive_params.ikl	-7160.29980
drive_params.baseline	0.43889
drive_params.right_joint_index	1
drive_params.left_joint_index	0
drive_params.max_translational_velocity	1.00000
drive_params.max_translational_acceleration	3.00000
drive_params.max_translational_brake	4.00000
drive_params.max_rotational_velocity	2.00000
drive_params.max_rotational_acceleration	15.00000

drive_params.ikr		SET
drive_params.ikl		SET
drive_params.baseline		SET
drive_params.right_joint_index		SET
drive_params.left_joint_index		SET
drive_params.max_translational_velocity		SET
drive_params.max_translational_acceleration		SET
drive_params.max_translational_brake		SET
drive_params.max_rotational_velocity		SET
drive_params.max_rotational_acceleration		SET

message.message	Ready

STOP | SEND | STORE | FETCH | REQUEST

drive_control.translational_velocity		SET
drive_control.rotational_velocity		SET

STOP | SEND

Client Interface (C library)

```
// opaque object
struct OrazioClient;

// opens connection
struct OrazioClient* OrazioClient_init(const char* device, uint32_t baudrate);

// destroyes orazio and close connection
void OrazioClient_destroy(struct OrazioClient* cl);

// sends a packet, the header should be compiled to match the packet type and size
PacketStatus OrazioClient_sendPacket(struct OrazioClient* cl, PacketHeader* p, int timeout);

// retrieves a packet, the header should be compiled to match the packet type and size
// the rest is filled
PacketStatus OrazioClient_get(struct OrazioClient* cl, PacketHeader* dest);

// how many motors is this firmware supporting?
uint8_t OrazioClient_numJoints(struct OrazioClient* cl);

// 1 cycle of handshake, advances the epoch
PacketStatus OrazioClient_sync(struct OrazioClient* cl, int cycles);

// queries all parameters (to be done after syncing)
PacketStatus OrazioClient_readConfiguration(struct OrazioClient* cl, int timeout);
```

Minimal C program to connect to the robot

- you find it here(sorry, the link is long)

  https://gitlab.com/srrg-software/srrg2_orazio/-/blob/master/srrg2_orazio/src/host_test/orazio_client_test_simplified.c