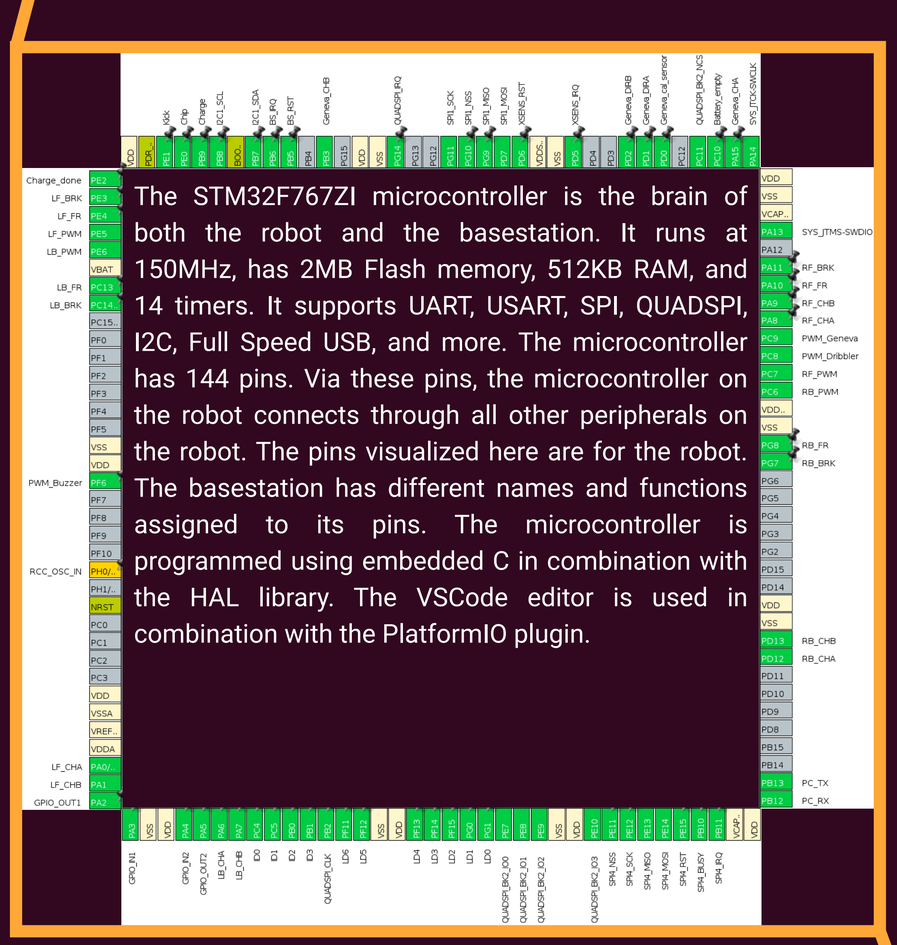
Embedded Overview of the Topboard

STM32F767ZI

Hardware timers

Timer	Frequency	Responsible for
htim7	100 Hz	Internal state and weels
htim10	1 Hz	Charging the capacitor
htim11	Variable	Buzzer



The STM32F767ZI microcontroller is the brain of both the robot and the basestation. It runs at 150MHz, has 2MB Flash memory, 512KB RAM, and 14 timers. It supports UART, USART, SPI, QUADSPI, I2C, Full Speed USB, and more. The microcontroller has 144 pins. Via these pins, the microcontroller on the robot connects through all other peripherals on the robot. The pins visualized here are for the robot. The basestation has different names and functions assigned to its pins. The microcontroller is programmed using embedded C in combination with the HAL library. The VSCode editor is used in combination with the PlatformIO plugin.

RoboTeam Embedded Messages

The RoboTeam Embedded Messages repository holds the definition of all packets sent between the computer, the basestation, and the robot. All packets come in two formats: A bit array, optimized for size (as seen on the right), and a C struct for ease of use. Functions are provided to easily switch between the two formats. Each type of packet comes with a unique header. The packet header, combined with the known packet lengths, allows sending multiple packets in a single transmission. Python bindings are provided to control the robots via python scripts. Python scripts are provided to automatically generate .c and .h files. No more manual bitshifting!

RobotCommand*

```
[---0--] [---1--] [---2--] [---3--] [---4--] [---5--] [---6--] [---7--] [---8--] [---9--] [--10--]
11111111 -------- -------- -------- -------- -------- -------- -------- -------- -------- -------- header
-------- 1111---- -------- -------- -------- -------- -------- -------- -------- -------- -------- id
-------- ----1--- -------- -------- -------- -------- -------- -------- -------- -------- -------- doKick
-------- -----1-- -------- -------- -------- -------- -------- -------- -------- -------- -------- doChip
-------- ------1- -------- -------- -------- -------- -------- -------- -------- -------- -------- doForce
-------- -------1 -------- -------- -------- -------- -------- -------- -------- -------- -------- useCameraAngle
-------- -------- 11111111 11111111 -------- -------- -------- -------- -------- -------- -------- rho
-------- -------- -------- -------- 11111111 11111111 -------- -------- -------- -------- -------- theta
-------- -------- -------- -------- -------- -------- 11111111 11111111 -------- -------- -------- angle
-------- -------- -------- -------- -------- -------- -------- -------- 11111111 11111111 -------- cameraAngle
-------- -------- -------- -------- -------- -------- -------- -------- -------- -------- 111----- dribbler
-------- -------- -------- -------- -------- -------- -------- -------- -------- -------- ---111-- kickChipPower
-------- -------- -------- -------- -------- -------- -------- -------- -------- -------- -----1-- angularControl
-------- -------- -------- -------- -------- -------- -------- -------- -------- -------- -------1 feedback
```

RobotFeedback*

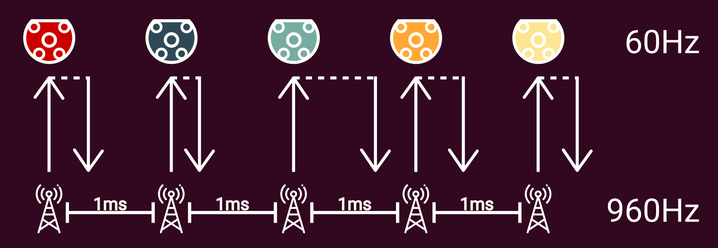
```
[---0--] [---1--] [---2--] [---3--] [---4--] [---5--] [---6--] [---7--] [---8--] [---9--] [--10--]
11111111 -------- -------- -------- -------- -------- -------- -------- -------- -------- -------- header
-------- 1111---- -------- -------- -------- -------- -------- -------- -------- -------- -------- id
-------- ----1111 -------- -------- -------- -------- -------- -------- -------- -------- -------- batteryLevel
-------- -------- 1------- -------- -------- -------- -------- -------- -------- -------- -------- XsensCalibrated
-------- -------- -1------ -------- -------- -------- -------- -------- -------- -------- -------- ballSensorWorking
-------- -------- --1----- -------- -------- -------- -------- -------- -------- -------- -------- hasBall
-------- -------- ---1---- -------- -------- -------- -------- -------- -------- -------- -------- capacitorCharged
-------- -------- ----1111 -------- -------- -------- -------- -------- -------- -------- -------- ballPos
-------- -------- -------- 11111111 11111111 -------- -------- -------- -------- -------- -------- rho
-------- -------- -------- -------- -------- 11111111 11111111 -------- -------- -------- -------- theta
-------- -------- -------- -------- -------- -------- -------- 11111111 11111111 -------- -------- angle
-------- -------- -------- -------- -------- -------- -------- -------- -------- 1111---- -------- wheelLocked
-------- -------- -------- -------- -------- -------- -------- -------- -------- ----1111 -------- wheelBraking
-------- -------- -------- -------- -------- -------- -------- -------- -------- -------- 1111---- rssi
```

* under development

TDMA Protocol

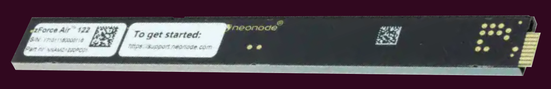
The robots can not all communicate with the basestation simultaneously. It would be as if eleven players talked to the coach all at once. The basestation is only able to communicate with one robot at a time. This only works if each robot waits for its turn to speak. This protocol, in which each robot has a specific timeslot to communicate, is called Time Division Multiple Access (TDMA).

As seen on the right, approximately every 1ms the basestation sends a command to a robot. The robot has around 0.5ms to respond with its feedback. Given a maximum of 16 robots, an approximate 1ms interval gives around 60 packets per robot per second.



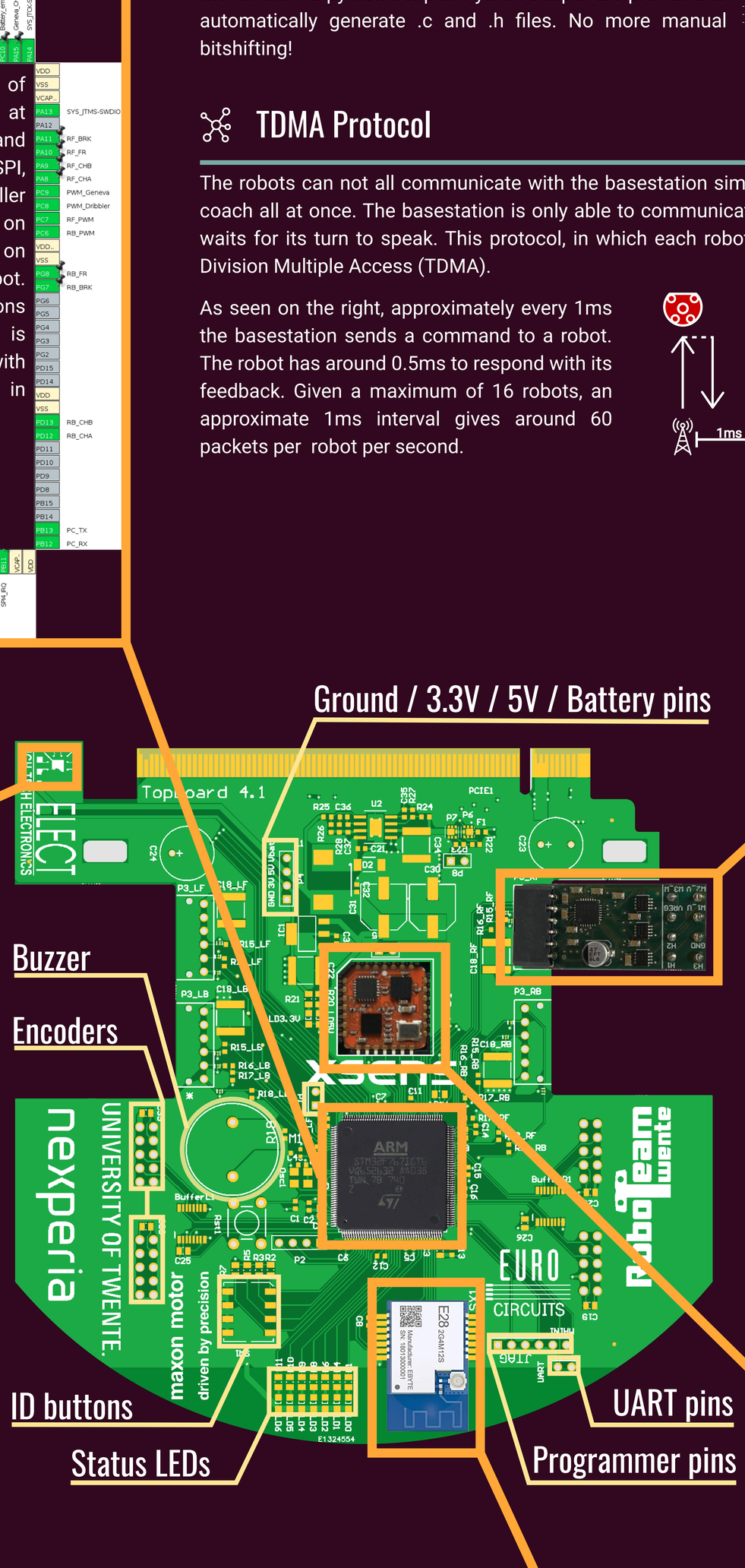
Ball Sensor

The computer requires about 100ms to decide the next action for each robot. Within this time, a ball can move up to 65 centimeters per the official RoboCup rules. The computer is too slow to send a kick command fast enough for the robot to kick balls moving at high speeds. To allow the robot the to kick a ball instantly, a ball sensor has been added. The ball sensor is a Neonode zForce AIR touch bar. It is placed at the front, aiming at where the robot would be able to kick the ball. The ball sensor scans at a rate of 200Hz. Between two scans, a ball can move at most 3.25 centimeters. The kicker has a width of 4.5 centimeters, thus the ball will always be in range. The computer can instruct the robot to anticipate the ball, and shoot it as soons as it is detected. The ballsensor allows for faster plays, which are harder to anticipate and defend against.



Motion Control

All of the major components on the topboard are combined in the motion control of the robot. The C program that runs on the microcontroller converts commands received by the SX1280 chip into direct instructions for the actuators. For one of these actuators, the wheel motors, this conversion requires some computational effort. The current state of the robot (velocity in angular and both lateral directions) is first estimated by combining wheel speeds from the wheel encoders and acceleration and angular velocity from the Xsens chip in a Kalman filter. This estimated state is then compared to the received commands and the difference is passed through a PID algorithm that produces the instructions for the motors to match the velocity commands as closely as possible.



Motor Drivers & Encoders

A motor driver controls the speed of a motor. The motor drivers receive a 25KHz PWM signal. A 100% dutycycle indicates no movement, and a 0% dutycycle indicates full power. A second signal is used to indicate the direction of rotation. A third signal indicates if the brakes should be applied. Each of our four drivers control a 50 Watt Maxon motor at 24 Volts. Each motor has an encoder, used to accurately measure the wheel speed and direction. These encoders are on a completely separate circuit, not connected to the motor drivers. An encoder provides 1024 ticks per rotation, giving a resolution of approximately 0.35 degrees per tick.

Xsens Chip

The Xsens Inertial Measurement Unit (IMU) is responsible for measuring acceleration, rotational velocity, and current direction. The IMU provides data at 100Hz. This frequency would not give enough data to properly estimate the robot state, were it not that much of the filtering and processing is already done by the IMU. The IMU has it's own frame of reference, initialized at boot time. This frame can deviate from the computer its frame, which takes the width and height of the field as x-, and y-axis. The computer sends camera data to the robot for calculating this deviation. The data provided by the IMU allows the robot to accurately execute commands from the computer. Instead of completely relying on the relatively slow feedback of the cameras, the robot can use the IMU data to precisely control its movements. The data of the camera can then be used to calibrate the drift in the IMU that builds up over time.

SX1280 2.4GHz chip

The SX1280 is used to wirelessly communicate between robots and the basestation. The basestation has two SX1280's, one for transmitting to the robot at 2.385GHz, and one for receiving from the robot at 2.395GHz. The robot has one SX1280 for both transmitting and receiving. At 2.4GHz, it can transmit up to 1.3Mbps in FLRC mode. It can send and receive packets of variable length up to 127 bytes.