**4. Application Development**

It is needed to have a software basis for ITU-AGVs that will be used to develop specific applications for various tasks in the future possible projects, theses and works. ITU-AGVs have built before ROS was developed. At the time when ITU-AGVs built, the software systems that used were different and custom so the software of the robots was written concerning them which became out-of-date now.

As mentioned previously in the Introduction chapter, the goal is to construct a set of applications for the basic problems and needs using up-to-date tools. It is desired to write the embedded code for LLPL so the robots can be communicate with ROS and to develop ROS applications for tele-operation, sensor integration and reading, odometry estimation, data collection and offline map building. With realization of this basis, ITU-AGVs can be used as multi-purpose indoor land vehicle kits available using rapidly for educational purposes, theses, autonomous system design and algorithm development at ITU Robotics Laboratory.

The processing work is divided with a hierarchy. Low Level Processing Layer (LLPL) is responsible for getting commands, communicating motor drivers to drive the motors as desired in the given commands, requesting encoder values and sending them to High Level Processing Layer. High Level Processing Layer is responsible for complex calculations and is the part where the ROS runs. The project is started with LLPL work and then HLPL after.

**4.1 Embedded Program Dev**

The embedded software for TMS320F28335 microcontrollers at LLPL is developed using Simulink Embedded Target Coder in MATLAB r2012b and then the make files are downloaded using TI Code Composer Studio v4.

**4.1.1 Com with EPOS**

The Maxon EPOS 70/10 motor drivers are designed to be used with CANOpen protocol. In CAN communication several hardware are connected as slaves to a master using a CAN Bus. The communication and configuration occurs with using array of variables called objects. Object dictionary includes all object addresses with 16-bit index and 8-bit sub index.

EPOS drivers have their configurable controllers and there are several driving modes such as position mode, velocity mode, profile velocity mode and so on. It is desired to send velocity commands to the LLPL and to settle the motors on the desired velocity references. So the operation mode would be selected as the profile velocity mode.

According to the EPOS 70/10 Manual {eposManual}, the motors are controlled with given profile velocity and acceleration limits and selection of motion profile type. Motion profile type can be selected as linear or sinusoidal. In the manual, all configuration and communication object values and their places in the work flow are provided.

In Simulink, a state chart diagram is created in order to make the software flow as a state machine. In the state chart, according to the EPOS 70/10 Manual the necessary configurations are being made. First, all CANOpen nodes are being reset and all slaves are being set as operational. Then according to the chart in Figure{eposProfileVelocity}, the operation mode is being selected as profile velocity mode, maximum profile velocity, profile acceleration, profile deceleration, quick stop deceleration and motion profile type values are being configured over their objects and a necessary reset is being done. After these configurations, the program enters a loop. In the loop, the commands are being taken from HLPL over SCI-A (Serial Communication Interface - A) serially. The commands are being parsed and left and right motor commands are separated and set as target velocity values. The control word is being sent and encoder values are being requested. After the encoder values are received, they are being sent to HLPL over SCI-B and the loop begins again. The flow of the state chart diagram in Simulink can be seen in Figure{simulinkStateFlow}.

**4.1.2 Com with HLPL**

The velocity commands are designed to be in one sixteenth of desired rpm values at the motor shaft before gear-box. LLPL takes commands in 16-bit integers. The command word will be sent starting with “#” character and ending with “!” character. The first 16-bits after starting character will be the left motor command and the second 16-bits until the end character will be the right motor command. Commands are 16-bits and the first byte of each command is for direction and the second byte is for one sixteenth of the rpm value desired at the motor shaft before the gear-box. Before sending the values to the drivers, this value is multiplied by. It is important to remind the gear-box on the motor since it reduces the rpm with a ratio of 1:100.

For example, if it is wanted to drive the wheels at 40 rpm, a basic calculation can be made. The rpm value at the shaft of the motor before the gear-box would be 40\*100=4000. This is the target rpm value, so the second byte of the command must have the value of 4000/16=250.

The direction is set such that, if the value of the direction byte is less than or equal to 127 it is counted as positive direction and the otherwise is negative direction. So the necessary word needed to be send to LLPL in order to drive wheels at 40 rpm in the positive direction should be;

#0 250 0 250!. The parsing is made in then made in LLPL.

EPOS 70/10 can provide various calculations with encoder values and it can give position and velocity. The encoder values are being sent in 16-bits to HLPL. Both SCI-A and SCI-B serial communications are set at 115200 baud.

**4.1.3**

In order to test the embedded software and the serial communication a simple test script is written with Python. In the script the direction and desired wheel rpm values are requested from the user for left and right wheels and the command word is calculated and sent over serial port to the ITU-AGVs. After using this test script, it is concluded that the LLPL is functionally working and the project can be moved on to HLPL.