2602 Core Library

Version 1.1

Albert Li

For Team 2602 only

Introduction

This Core Library is a series of ROBOTC C code functions and data structures that contains basic and advanced functions using in VEX EDR competition. The library is distributed as a single header file that is included at the top of the main ROBOTC source file by means of an #include statement. The library provides the following functionality.

- Setup gyro(s)
- Filter gyro data to eliminate noises
- P control turn based on gyro read
- PID control for chassis
- Automatically close the pincher based on speed calculation
- P control open pincher
- P control calculartor

Installation

The Core Library for 2602 is included in the main ROBOTC source file as follows

```
//*!!Code automatically generated by 'ROBOTC' configuration wizard
!!*//

//include library

#include "SmartMotorLib.c"

task main()

{

//user code

}
```

The Core Library should be located in the same directory as the main source file.

How to use the library

The general procedure for using the Core Library is as follows.

- Initialize the library
- Call functions and task when needed

Initialization

The Core Library is initialized by calling following functions

```
void setArm(tMotor motorR, tMotor motorL, tSensors port, int down, int
full,int none,int star,int stars,int cube)
```

This function initializes the arm part of the library. User should enter the port numbers of right and left motors of the arm, the port of encoder of the arm, and the critical encoder value when arm is down, up, hold nothing, hold a star, hold 3 stars and hold cube, respectively.

```
void setChassis(char side,tMotor motorport, tSensors sensorport, float
kp, float ki, float kd, float integrallimit)
```

This function initializes the chassis part of the library. User should first enter the side they want to initializes: 0==Right, 1==Left. User then should enter motor port(two motors should be Y-Cabled), encoder port, P-value, I-value, D-value and integral limit for PID control.

```
void setPincher(char side,tMotor motorport, tSensors sensorport,int openMV,int midMV,int closeMV)
```

This function initializes the pincher part of the library. User should first enter the side they want to initializes: 0==Right, 1==Left. User then should enter motor port, potentiometer port, critical potentiometer value when pincher is fully open, in the middle, and fully closed, respectively.

```
void setGyro(tSensors port,int scale)
```

This function initializes gyro. User should enter the analog port of a tuned gyro, and the sensor scale of such tuned gyro.

Chassis Movement

```
void chassisPID(bool forward, bool ifLift, bool ifHoldPincher, int target)
```

This moves chassis straight for `target` encoder counts.

If `forward` is set to `true`, chassis moves forward. Other wise, it moves backward.

If `ifLift` is set to `true`, arm lifts when the chassis is close to `target`.

If `ifHoldPincher` is set to `true`, pincher uses motor power to hold while moving straight.

```
void gyroTurn(int nDegree,int timeLimit)
```

This function turns the robot to nDegree within timeLimit using P-control.

Usually the timeLimitis 1000ms.

nDegreeis a field center value. It is 0 when the gyro is setup; it increases when the robot turns anti-clockwise; it decreases when the robot turns clock-wise.

Arm Movement

```
void autoArmDown()
```

This moves the arm down using a while loop based on critical encoder value entered during void setArm.

```
void autoArmUp()
```

This moves the arm up to full using a while loop based on critical encoder value entered during void setArm.

```
void autoArmHold(char type)
```

This moves the arm to the hold position and the rubber bands would hold at the position.

User should use parameter type to choose holding model based on what is inside the pincher.

type 0==pincher holding nothing

type 1==pincher holding 1 or 2 star(s)

type 2==pincher holding a cube

type 3==pincher holding 3 stars

Pincher Movement

```
void autoOpenPincher()
```

This fully opens the pincher based on a critical potentiometer value entered in setupPincher. It should only be used in autonomous.

```
void autoOpenPincher()
```

This closes the pincher based on speed calculation. It should only be used in autonomous.

```
task openPincher()
```

This fully opens the pincher based on a critical potentiometer value entered in setupPincher. It should only be used in driver control.

```
task closePincher()
```

This closes the pincher based on speed calculation. It should only be used in driver control.

Sensor Reset and Gyro

```
void resetChassisEncoders()
```

This resets two encoders of the chassis.

```
void resetArmEncoder()
```

This resets the encoder of the arm.

```
void gyroSetup()
```

This initializes the gyro. It takes about 3 secs. The robot should be absolutely still during setting up.

```
task gyroFilter()
```

This is a background task that filter out the gyro noises. This should be started after setting up the gyro. Usually, this should only be used in autonomous.