Positioning Software Documentation

Basic GPIO Library: WiringPi

Main functions used from wiringPi:
digitalRead(pin);
digitalWrite(pin);

MotorConfig.cpp

Instance Modifier:

MotorConfig(): Using the instance modifier with no arguments (i.e. MotorConfig leftmotor;) will initialize all global variables, but will not set the hardware pins as there are no arguments. If this is used (which it currently is in the EndEffectorConfig instance modifier) then the set function "setHardware" must be used after initializing the MotorConfig object (i.e leftmotor.setHardware(1,0,2,3);

MotorConfig(int, int, int, int): This instance modifier will initialize all global variables and configure hardware pins in the order (step pin, dir pin, outer switch, inner switch).

Custom Functions:

setHardware(int step, int dir, int out, int in): used to assign pins for limit switches and motor controller pins

setSpeed(float speed) ; use value from -160mm/s to 160mm/s (float) to set the motor speed. + speed = drive inwards and - speed = drive outwards. (This is slightly confusing because coordinate frame of the motor calls the inner switch position '0' in the end effector class). setSpeed is designed to be a subscriber to motor speed commands, which could be changed at any rate (even once a second if you wanted).

motorDrive(): this directly calls digitalWrite at the specified setSpeed() to toggle the step pin for the motor being configured. This function is used inside controlLoop() when motor driving is desired. Note: if setSpeed(0), then the conditional statement of motorDrive() cannot be entered and therefore pin cannot be toggled, and motors will stop.

controlLoop() : looking at the limit switches, control loop determines state of motor (run, single inhibited, double inhibited) and reports that to the public variable driveState. If motor is in a good state (no switches pressed), it calls the motorDrive() function. The control loop function needs to be implemented in a high frequency loop (preferably separate ROS node) which should be left on at all times that motors could possibly accept speed commands.

accToSpeed(): this is also meant to be used as a subscriber to motor speed commands, and use a time based acceleration to achieve the desired speed, which can be changed outside of the class (acceleration variable).

setGoalPosition(): to be called in conjunction with goalPosition() function. This set functions assigns the appropriate global variables (such as desired steps and direction) to be used by goalPosition. The reason for a global variable in this case is that we need memory of the number of steps to begin with in order to properly apply acceleration and deceleration curves properly and not through timing.

goalPosition() : to preface; never run this function at the same time as controlLoop() — otherwise bad things will happen as the function actually calls controlLoop within it. So for example, during visual servoing control loop should be in a while loop with setSpeed subscribing to commands sent. But during go home, control loop should not be in a while loop or called in any other context. I assume this will be easily implemented with a state machine.

Important Variables:

deadBandSpeed : this determines cutoff in set speed function, if within 0-abs(deadband), then the set speed will default to 0 speed.

debounceTime : time required to determine proper button position (to be tested with new logic implemented in controlLoop());

goalAcceleration : previously called defaultAcceleration, this defines acceleration in mm/s/step used in the internal goalPosition function. Different than acceleration value which is in mm/s/s. The reason for this is so that deceleration is not time based and can end at 0 speed precisely at a specific step.

currentSpeed (mm/s): current set speed of the motor. And should physically represent what's going on

currentDelay (ns): current delay time to achieve currentSpeed.

motorDir : direction of motor (1 = inward , -1 = outward)

driveState : current state of motor inhibit :: (1 = drive | 2 = inner inhibit |. -1 = outer inhibit. | 0 = both inhibit) the motors can still drive if in states 2 or -1, but only in the opposite direction of the inhibit.

EndEffectorConfig.cpp

Instance Modifier:

EndEffectorConfig(int left, int right): Using the instance modifier with 2 integer arguments, this will initialize necessary global variables and set the hardware pins for left and right motors. The integer arguments are passed to initialize the current step count of the left and right motors , which was sometimes helpful in testing but now with calibrateZero, these arguments will quickly be reset after calibration.

Custom Functions:

updateMotorPosition(): updates the position of left and right motors and assigns them to global variables. Inner switch is position 0, outer switch is position +250mm

updateCurrentPosition(): updates end effector current position (x,z) also stored as global variables

goToPosition(x, z, speed): using the goalPosition motor functions, this function actuates end effector to desired user position at the speed specified (will automatically call both setGoalPosition and goalPosition in MotorConfig)

calibrateZero(speed) : calibrates motor position, but MUST calibrate when mechanism relatively folded to ping outer switches as calibration points (this is faster and simpler). Tell me to add logic to include inner switch calibration as well if it seems like there is a desire to calibrate whilst extended.

moveInX(speed) and moveInZ(speed) : both of these are used for actuating along a single axis. While MotorConfig::controlLoop() is currently called in these functions, please tell me if ROS node will be set to run controlLoop() independently because this function will then need to be modified.

directionalDrive(CMD X, CMD Z) : using IK , we are able to drive at a desired end effector X and Z speed (mm/s) but only limited to 25mm/s in X and 100 mm/s in Z. This will be more than sufficient if an initial rough estimate position brings the EE out quickly in Z to a +/- 1 cm x position relative to vine. To be clear, this is for simultaneous X and Z actuation.

directionalDriveMAG(CMD X, CMD Z, speed): old function that uses a heading and set speed to travel at specified unit vectors at specified speed

Important Variables

```
float leftMotorPosition; // left motor Position in mm
float rightMotorPosition; // right motor Position in mm

float rightMotorSpeed; // right speed in %
float leftMotorSpeed; // left speed in %

float xPosition; // end effector position in mm (relative to center of macron)
float zPosition; // end effeector position in mm
```

Bool CalibrationSuccess; 1 if robot is calibrated, 0 if it is not….. can use. If new calibration is requested mid run, will need to set this value to 0 before calling calibrateZero.