Spherical Tumbler v1.0

Professor Lynch and Professor Lueptow

Scott McLeod

June 11, 2008

Contents

Quick Start

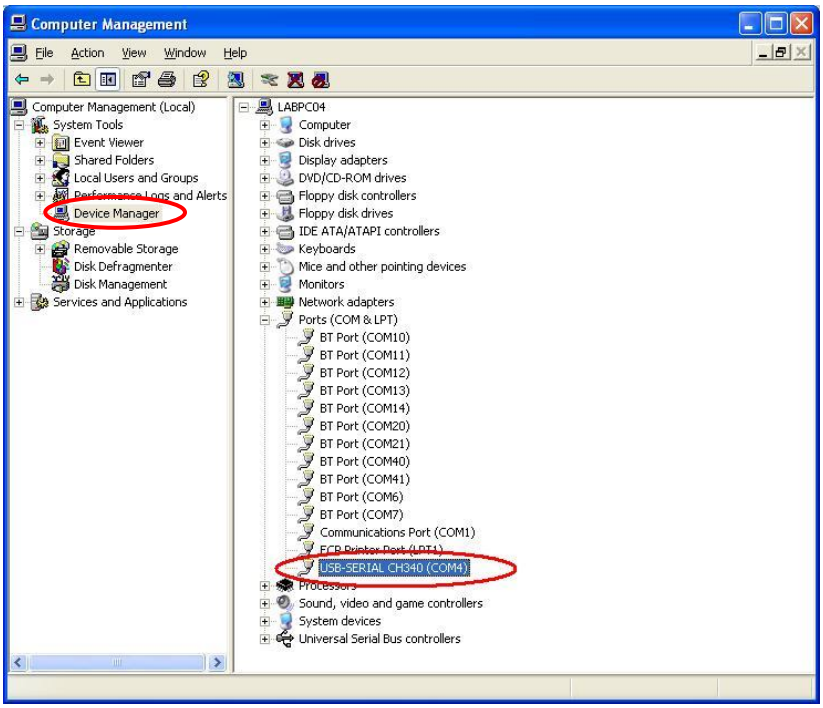
Install USB - Serial Drivers

The communication between the PIC and the PC is done with a USB to TTL serial converter cable. To use this cable, the drivers must be installed first. They are included in the software package, and are located at the following website: http://www.ftdichip.com/Drivers/CDM/CDM%202.04.06.exe. Follow the instructions to install this driver software.

Determine the COM Port

To open the correct COM in MATLAB, you will have to input a number corresponding to the serial port connected to the spherical tumbler. Follow the instructions below to identify this device (windows only).

1. Right click on “My Computer” and select “Manage”.
2. Select the “Device Manager” from the left hand column.
3. Expand the group titled “Ports (COM & LPT)” as shown in the picture below.



4. Look for the device labeled “USB-Serial”. Note the COM# that corresponds to this device. *Note if this name doesn’t appear try unplugging the USB cable, and seeing if any devices disappear. If a device doesn’t reappear when the USB cable is plugged in again, you may have to reinstall the driver software.

Connecting the System

1. First Plug in the USB end of the USB->serial port into the computer and identify the COM port as detailed previously.
2. Connect the other end of the serial cable with the black wire towards the top of the circuit closest to the PICs.
3. Connect the 5V power supply to the wall and into the banana plug adapter on the right side of the circuit board. **Note this is the same plug as the master PIC, but it is moved off to avoid interfering with the tumbler's motion.
4. Plug in the 12V power supply to enable the motors. It at any point the motors perform undesired actions, unplug this power supply and cycle power to the 5V logic.
5. Now you are ready to connect to the tumbler via MATLAB.

Important Cautions

There are two limitations with the current implementation of the controller design. Both of these must be taken into consideration to protect the tumbler.

1. Pauses MUST be added between subsequent position commands. The controller will NOT wait or pause between motions and will move indeterminately to the final position command. Both axes may be commanded at the same time, but two sequential position commands on the same axis must be separated. The time for this pause can be performed in MATLAB by keeping track of the current position, the new position, and the current velocity. For example:
   Pause(abs(newPosition – oldPosition)/(360 * velocity) * 60);
   oldPosition = newPosition;

2. The controller currently has no failsafe to prevent the axis motor from spinning too far. Since the motors are hard wired to the circuitry, spinning the axis motor beyond a certain point will physically pull the circuitry. To prevent this from happening it is recommended that a maximum angle for the axis is set. Be sure to set the home position correctly when powering on the tumbler. If it goes outside the mechanical range, the best option is to unplug the 12 power supply and then the 5V logic.

MATLAB Setup

Set the current directory to the MATLAB folder of the software package ("\Tumbler\MATLAB\"). Now you can either control the tumbler from a graphical user interface (GUI) or via a scripted ".m" file.

Scripted Mode

To run a script, simply save your script file in the ("\Tumbler\MATLAB\") directory. A sample script file with comments is included in this directory and on the following pages. The scripted mode is more useful for running more complex or repetitive motion commands.

The only change needed in this code will be specifying the serial port (variable COM_PORT) at line 6. The command MotorControllerConnect(COM_PORT) uses this parameter to open the corresponding serial port.

Once the COM_PORT variable is appropriately set, you can adjust the velocity and change the position commands as desired. Note that all velocities are passed as positive numbers (even for backwards motion) and need only be set once. The units for velocity are RPM, and the units for position commands are degrees (as computed with the 5.5" sphere). The position commands move the sphere directly to the final position at the set velocity. Computations/functions for motion related to angle of repose must be generated in MATLAB manually (i.e. move to 120^{o}, pause, move to 90^{o}).

The following pages show the included simple MATLAB script for controlling the spherical tumbler.

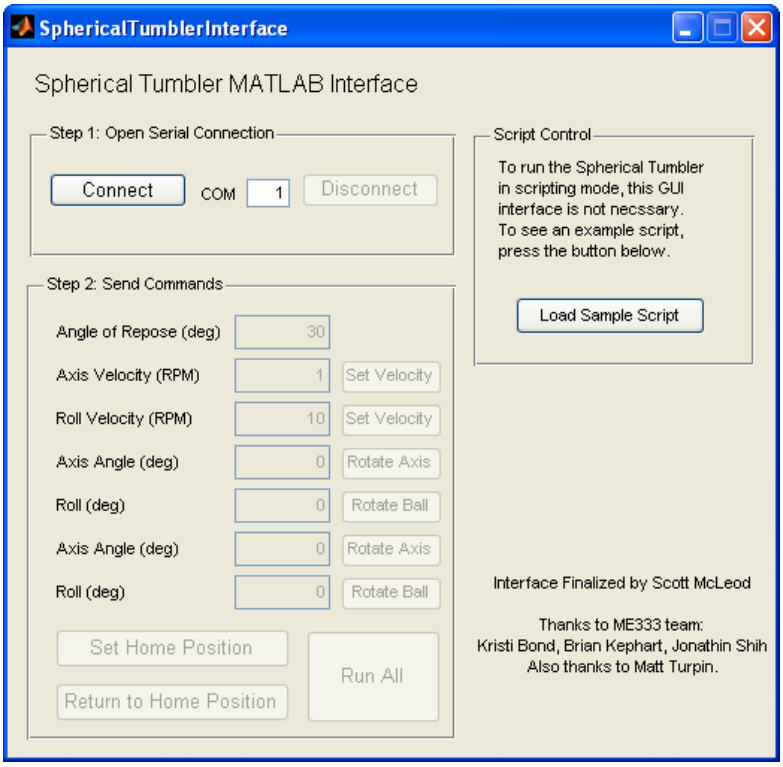
Scripted Code Example

```
% Spherical Tumbler MATLAB Interface
% This is a sample program that demonstrates how to script the tumbler
% Written by Scott McLeod - June 9, 2009

% COM Port Setting
COM_PORT = 15;

% Output time so we can distinguish data/runs in MATLAB Command Window
clc
timeInfo = clock;
fprintf('Spherical Tumber MATLAB Interface\n');
fprintf('Date: %d/%d/%d   Time: %d:%d:%2.0f\n', timeInfo(2), timeInfo(3),
timeInfo(1), timeInfo(4), timeInfo(5), timeInfo(6))

% Attempt to open the serial port if not already connnected
if (~exist('mySerialObj', 'var'))
    mySerialObj = MotorControllerConnect(COM_PORT);
end

% Check to make sure the serial object is writable
if (~strcmp(mySerialObj.status, 'open'))
    fprintf('***** ERROR: Could not open serial port object\n');
    fclose(mySerialObj);
    delete(instrfind)
    clear mySerialObj
    return
end

% If we got here, serial port should be openend and writeable
fprintf('Serial Port Sucessfully Opened and Connected.\n');

% Set up the slave PIC addresses -- DO NOT CHANGE
AXIS_MOTOR_ID = 0;
ROLL_MOTOR_ID = 1;

% Set current position to home position
% Note in general, you don't want to reset the home position as this
% will cumulate error in the position tracking.  Instead, it is better
% to perform the positions all relative to a single starting location.
mc_resetpos(AXIS_MOTOR_ID, mySerialObj);
mc_resetpos(ROLL_MOTOR_ID, mySerialObj);

% Set motor velocities
% This operation only has to be set once, but may be changed whenever
% All velocities should be entered as positive numbers (even for backwards
% motion)
set_mc_velocity(1, AXIS_MOTOR_ID, mySerialObj);
set_mc_velocity(10, ROLL_MOTOR_ID, mySerialObj);



% Now we can change the desired program actions
```

```
% set_mc_position commmands are sent with inputs:
% (# degrees, motor_id, serial object)

old_axis_angle = 0;
old_roll_angle = 0;

for iteration = 1:5
    fprintf('Iteration %d.\n', iteration);

    % Move axis motor and pause appropriately
    axis_angle = -90;
    set_mc_position(axis_angle, AXIS_MOTOR_ID, mySerialObj)
    pause (abs(axis_angle - old_axis_angle)/(360 * 1) * 60 + 1);
    old_axis_angle = axis_angle;

    % Move roll motor and pause appropriately
    roll_angle = 90;
    set_mc_position(roll_angle, ROLL_MOTOR_ID, mySerialObj);
    pause (abs(roll_angle - old_roll_angle)/(360 * 10) * 60 + 1);
    old_roll_angle = roll_angle;

    % Move axis motor and pause appropriately
    axis_angle = 0;
    set_mc_position(axis_angle, AXIS_MOTOR_ID, mySerialObj)
    pause (abs(axis_angle - old_axis_angle)/(360 * 1) * 60 + 1);
    old_axis_angle = axis_angle;

    % Move roll motor and pause appropriately
    roll_angle = 0;
    set_mc_position(roll_angle, ROLL_MOTOR_ID, mySerialObj);
    pause (abs(roll_angle - old_roll_angle)/(360 * 10) * 60 + 1);
    old_roll_angle = roll_angle;
end


fprintf('Process Complete.\n');
```

GUI Mode

To launch the GUI, type SphericalTumblerInterface. The window as shown below will appear.



Here, we can see that the first thing we must do is set the COM port as identified in the previous step. Enter the integer corresponding to the COM port on your system. In the picture on the previous page, a 4 would be entered into this box. Once the COM port is specified, click the Connect button. If MATLAB was able to open the port successfully, the tumbler commands will become active. If MATLAB is unable to open the serial port, ensure the COM number is correct, and that no other programs are accessing that COM port. If MATLAB is still unable to connect, execute the command “delete(instrfind)” and “clear all” or restart MATLAB.

Note before closing the MATLAB Tumbler Interface GUI, you should click the Disconnect button to properly close the serial port.

To use the GUI, simply enter the numbers corresponding to the motions desired. The individual commands will send single commands, while the Run All button sends them all out in sequence (with computed delays based on distance moved). You should wait to send a single command at a time once the tumbler has reached its final position.

The velocities will be retained by the slave PICs, and need only be set once, but may be changed between position commands. The Set Home Position button will change the current position of the motors to the home position, (0^o, 0^o). To avoid cumulating error, you should avoid dynamically resetting the home position, and rather computer motions based on the single home position. (Instead of commanding a 90^o rotate, Set Home, command 90^o rotate; you should send the commands 90^o rotate, 180^o rotate).

This GUI interface will attempt to move the tumbler with the entered angle of repose. To perform a 90^o rotate on the roll axis, it will command a position of (90^o + repose), pause, and command a position of 90^o.

Lastly, the Return to Home button will put both motors back to their home position (0^o, 0^o). They will both move at the same time without repose information, so this will most likely not be useful for motion tumbling commands.

Electrical Design

To control the tumbler, three 18F4520 Microchip PIC microcontrollers are used. One of the PICs is used as a "Master" which controls its two "Slaves". The Master PIC is used to communicate to MATLAB and to control the slave PICs. The slave PICs then implement a basic algorithm to control a single brushed DC motor (one for each axis).

The master PIC is connected to the MATLAB interface via a serial connection (over a USB->serial cable). It then issues commands to the slave PICs via I^2C.

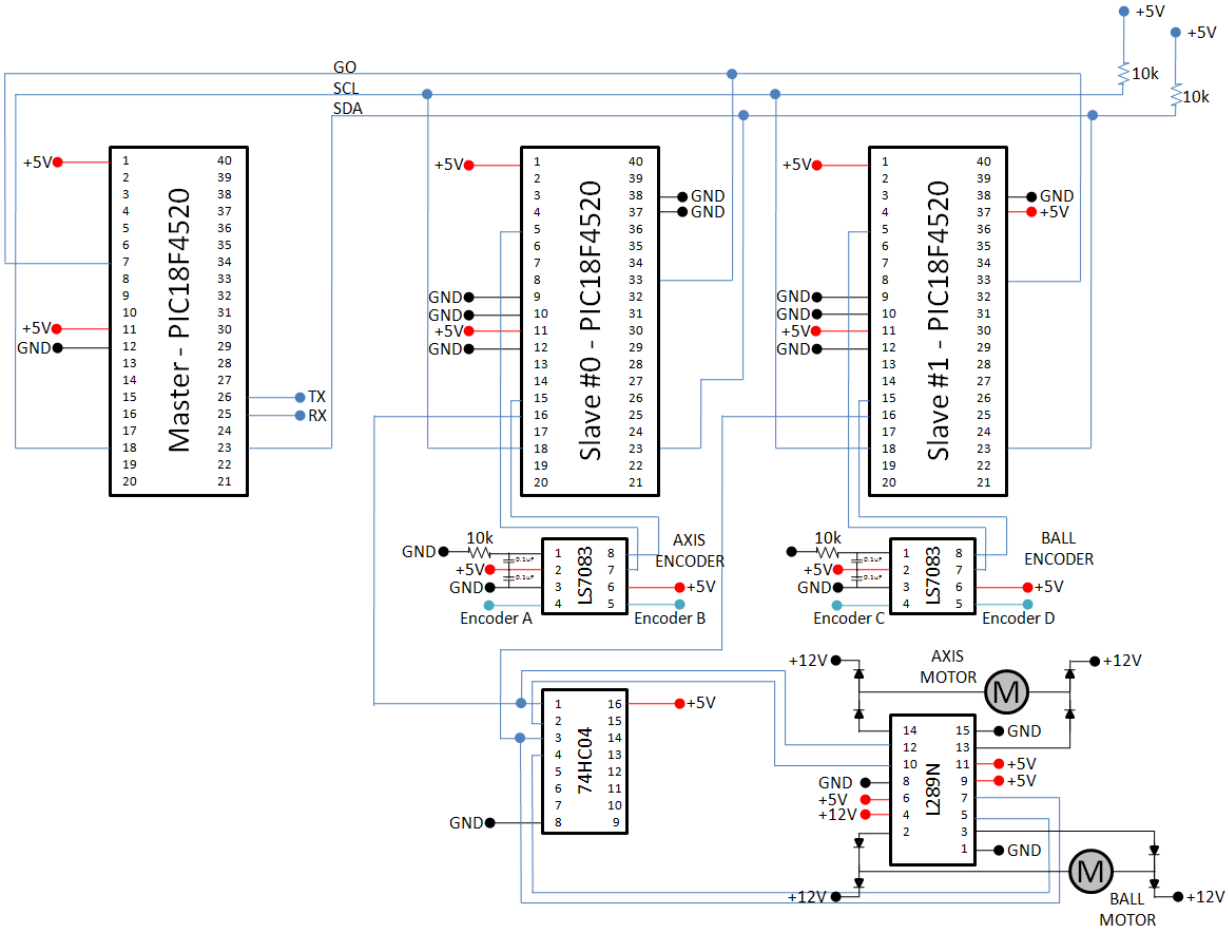
Two separate slaves are needed as each device has a hardware limitation of 2 external counters. Each motor has an optical two-channel encoder attached to its output shaft. As decoded by a quadrature decoding chip, this results in a resolution of 39000 encoder counts per gear head revolution. The motion of the ball and turn table is then computed based on a relative gear ratio based on their diameters. These measurements are shown below. The math for computation of degrees to encoder counts is performed in MATLAB (set_mc_position.m and set_mc_velocity.m).

Gearing/Coupling	**Diameter (in)**	**Ratio**
Sphere	5.5	2.62
Main Drive Wheel	2.1	
Turn Table	12	20.00
Table Drive Wheel	0.6	

Electrical Schematic

This file is located in the root folder of \Tumbler\Schematic.pdf

In this setup, slave #0 corresponds to the controller for the axis motor while slave #1 corresponds to the controller for the ball roll motor.

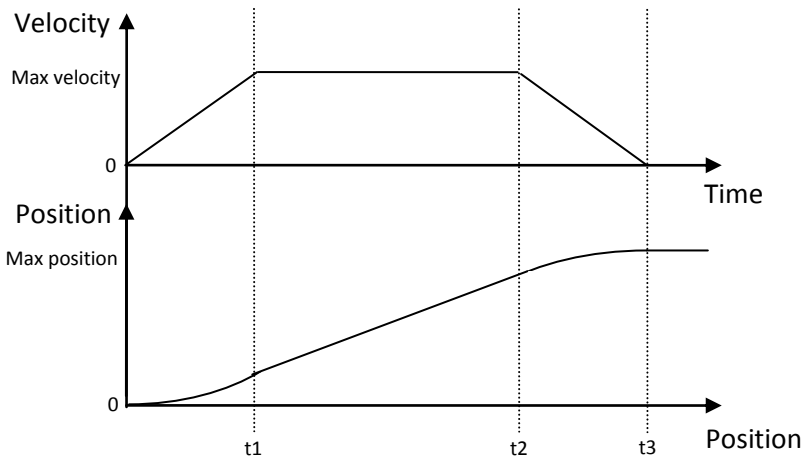


Controller Design

* The PIC code is included in the folder \Tumber\PIC_master and \Tumbler\PIC_slave.

The root of the controller implemented on each of the slave PICs was a proportional error controller. Running at 40MHz, the slave PICs sample the quadrature encoder counts at a rate of 2kHz (500µs period) as produced by the optical encoder on the motor output shaft. With such fine resolution, the proportional gain implemented is a unity gain. The output control from the PIC to the DC motors is a pulse-width-modulated signal. This PWM signal (and its inverted form) is used to drive a "full" H-bridge circuit. The internal hardware on the PIC is able to drive a PWM signal with 10 bits of resolution. In this sense, an error greater than 512 encoder counts results in complete saturation of the motors. At this sampling rate, the commanded positions achieved a quick response with no visible significant overshoot.

On top of this position controller, the commands sent from MATLAB are a velocity and final position. To perform this motion, a second control loop is used to move with a trapezoidal motion profile. In essence, the slave PIC receives a maximum velocity, final position and has an internal acceleration variable. Before executing the control, the PIC computes the times and increments corresponding to the three phases of the motion profile. These position commands are translated to a simple Euler integration resulting in a trapezoidal velocity profile. These phases are outlined below.



- Set acceleration, velocity, position to 0.
- If time < t1, increment velocity by acceleration. Increment position by velocity.
- If time < t2, set velocity to max velocity. Increment position by velocity.
- If time < t3, decrement velocity by acceleration. Increment position velocity.
- If time > t3, set position to final position.

Future Mechanical Improvements

Currently, the largest issues with the rotation of the tumbler are related to the motion of the sphere. In particular, the sphere tends to rotate with the axis motor and drift when rolling.

A first idea to add very low coefficient springs between the three ball castors and their back supports such that they would always remain in contact with the sphere. As the ball is not perfectly spherical, simply moving the ball castors inward over constrains the ball in certain orientations. By adding very light springs, the casters could theoretically remain in contact with the sphere, while allowing for slight variations in diameter due to the orientation of the ball.

Rotation with Axis

The first problem is related to an axial move of the drive motor on the rotating tray. When the tray is rotate, the ball tends to twist as the drive motor is still in contact with the ball. Some solutions we brainstormed were to add a new actuator to either lift the ball off the drive motor when rotating the axis, or to hold the ball in place via an actuated friction grip.

Imperfect Motion along Drive Wheel

A second problem is the motion of the sphere when commanded to roll in one direction. To fix this issue, one thought was to switch from a single centered drive wheel to two drive wheels connected to the same output shaft of the motor. This would create two points of contact, and decrease the likelihood for the ball to drive or “twist” sideways. Unfortunately, this would prevent motion of both axes at the same time, and would require a fix to prevent the axis motor from spinning the ball.