Autonomous Ping Pong Collection Robot

EECS 373, Intro. to Embedded System Design Project

Binhao Qin, Che Chen, Hanxi Wan, Yijia Gao

April 12, 2022

Table of Contents

Table of Contents

Overview

- Table Tennis Multi-ball Training
- Collecting balls is tiring and time-wasting
- Avoid body pain caused by frequently bending down

Main Features

- Collector: 1 servo, 3D printed components
- Motion: 4 mecanum wheels, motors (H-Bridge, PWM)
- CV: recognition & localization
- Inter-Controller UART Communication
- Manual mode: debugging and playing with N64 Controller
- Displaying statistics: LCD, I2C

Table of Contents

Collector Design

- roller with rubber bands to "catch" and reserve balls
- motor free, rolled by friction
- servo to add pressure or raise the roller up

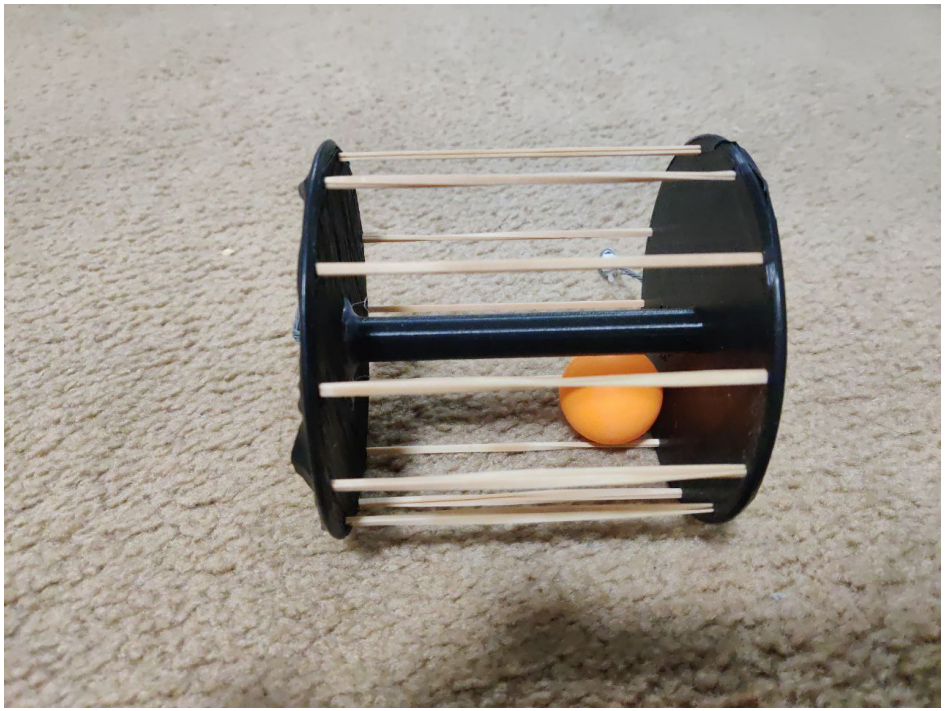
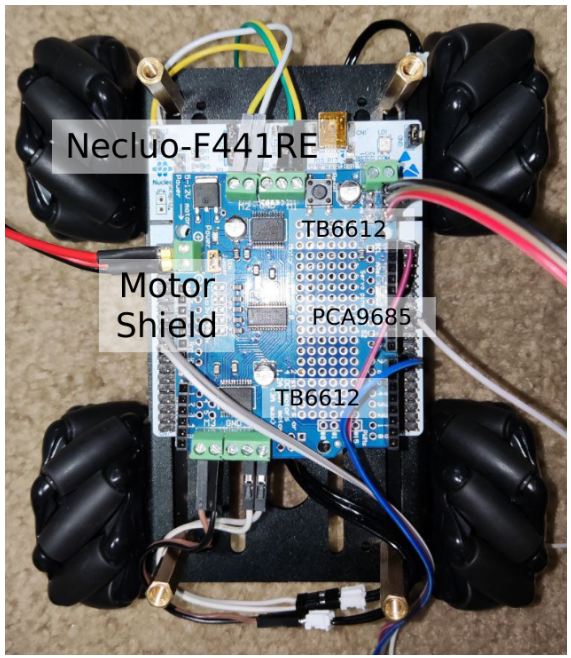


Figure: Roller with a ball

Motors and Servos

- Task
  - Control totally 4 motors and 1 servo
- Solution
  - Motor Shield
    - 1 * PCA9685 I2C PWM Controller
    - 2 * TB6612 DC drivers
  - Send command via I2C to set PWM Controller, which controls motor drivers and servos



Motors and Servos

- Challenge
  - Datasheets available for PWM Controller and DC drivers, but no datasheet for the Motor Shield
- Solution
  - Refer to Arduino library provided by MotorShield manufacturer
  - Use logic analyzer to debug I2C transmission

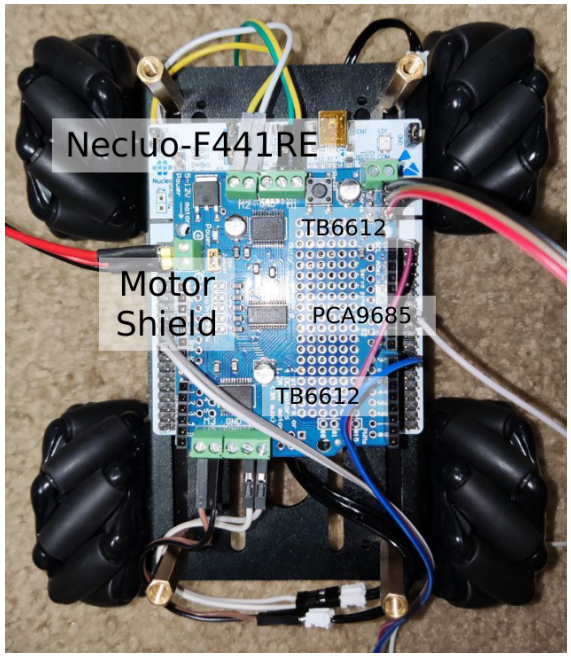


Table of Contents

Ball Recognition

- assume ping-pong ball to be yellow
- convert to HSV color space for easy color extraction
- extract a range of color from the frame that may be the ball
- find contours on the extracted mask
- for big contours, calculate the mininum enclosing circle to be the ball

Localization

- We know the actual radius of the ball R, focal length f, and size of the ball on the frame r
- The distance of the ball d can be calculated as $d = \frac{fR}{r}$
- Thus the position of the ball in camera coordinate can be calculated
- Match the balls with previous frame using ICP algorithm and get the camera motion

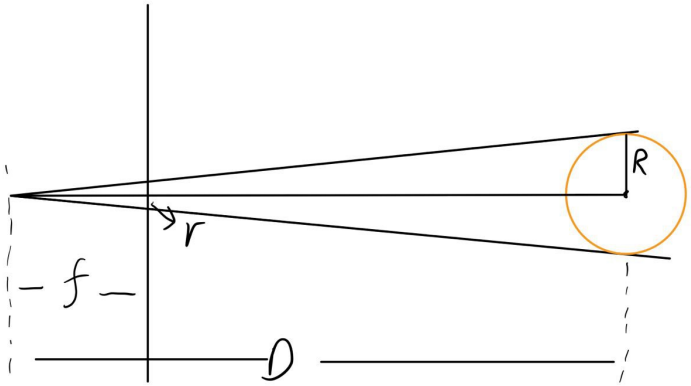


Table of Contents

Inter-Controller Communication

- Recognition and Localization computed on Raspberry Pi, need to give control information to STM32 board
- Serial UART communication is used
- Task: defining an efficient and stable communication protocol
- Solution:
  - 2 0xFF to start transaction
  - 1 Header byte to indicate device to operate
  - 1 or 2 data byte(s) depending on device type
  - 2 0xFF to end transaction
  - slave device transmit 1 byte to ensure completion

Table of Contents

N64 Controller Protocol

- 1-wire, half-duplex serial protocol
- OD/OC circuit with pull-up resistor required to avoid multi-driver issue
- 3-4 μs per bit transaction, and at least 2x sampling rate for receiver, which is sub-megahertz
- little endian but MSB first instead of LSB first

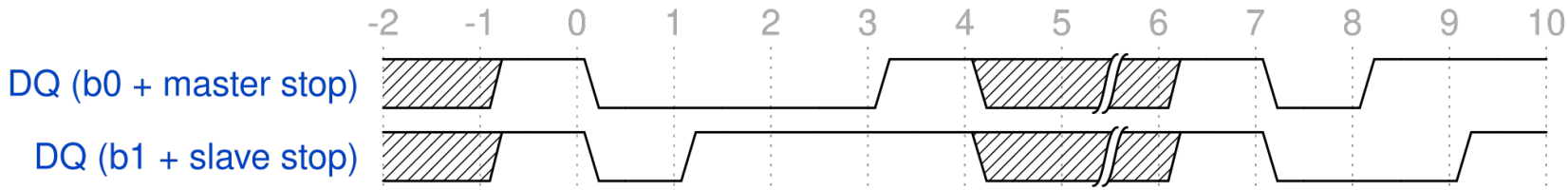


Figure: Timing Diagram of Bit Transaction

- **What is the major difficulty in implementation?**

Interfacing Challenges

- Welcome to real-time programming!
- **Time is precious resource!!!**
  - OD/OC design makes rising edge slow
  - internal pull-up resistor with 40kΩ is catastrophic
  - need microsecond level delay, but timer MMIO is slow
  - interrupts/buffering cannot be tolerated
  - memory accesses cannot be tolerated
  - function calls with stack pushes/branching cannot be tolerated
- Solutions:
  - use small external pull-up resistor (4.7kΩ)
  - implement microsecond delay by counting instructions
  - RAII CPU mutex locks interrupts for atomicity
  - use very few variables inside critical section
  - force function calls to be inline inside critical section

A New Issue: Mix Compiling C/C++

- As stated before, in order to meet the timing requirements, we would like to have RAII mutexes and inline functions.
- This cannot be easily achieved using C, and therefore C++ is used to implement these features.
- **However, C/C++ name function labels differently in assembly:**
  - C: `int f(int);` compiles to `f:`
  - C++: `int f(int);` compiles to `_Z1fi:`
  - Need `extern "C"` syntax to tell the compiler about C functions
- More on this topic: Check out name mangling on the Internet

Table of Contents

Setting Up LCD

- Challenges: sending 8-bit data/command through 4-bit channel
- Solutions:
  - EN falling edge sensitive, 2 transactions per byte
  - Cut the data in Upper 4-bit and Lower 4-bit
  - Configure data and send

```
void LCD::send(uint8_t byte, bool type)
{
        uint8_t encoded[4] = {0};
        uint8_t byteHigh = byte & 0xF0;
        uint8_t byteLow = (byte & 0x0F) << 4;
        encoded[0] = byteHigh | enableHigh | (type & 0b1);
        encoded[1] = byteHigh | enableLow | (type & 0b1);
        encoded[2] = byteLow | enableHigh | (type & 0b1);
        encoded[3] = byteLow | enableLow | (type & 0b1);
        HAL_I2C_Master_Transmit(&i2c, address, encoded, 4, HAL_MAX_DELAY);
}
```

Table of Contents

Next Step

- Assembly all the parts
- Set up simulation environment
- Conduct all-around testing

Thank you!

Q & A