What is Mechatronics?
Project Description
Autonomous system example
Course Organization
Huzzah32/ESP32 overview
FreeRTOS tasks and timing
What is Mechatronics?

- Moore’s Law for electronics
- Moore’s Law for mechanics(?)

Key Technologies for Mechatronics:
- Signal processing
- Control
- ...

Folded mirror array
Project Description

• Design Autonomous Race Car
• Unknown track (in principle, but home track will be known)
• Follow track without hitting cones.
• Stop at end of track.
• Winning speed: 3.3 m/sec (Spring 2017 Natcar winner)
• (2018 9.7 ft/sec = 3.0 m/sec)
• Learning allowed (though only have 5 minutes total for best run)
Hardware

You should have:
- Huzzah32
- RCNitro or equiv car
- USB Battery

To be shipped from Berkeley
- Line camera
- Jumper wires
- Analog Discovery scope
adapted from Thrun et al, JFR 23(9) 2006
adapted from Thrun et al, JFR 23(9) 2006
Course Organization

• Tues Lecture -> Wed Lab Demo -> 9 days -> Fri checkoff hour (live Zoom) (tba)
• Partners: 2 is good, 3 is possible.
• Return equipment at end of semester for grade, and car+CPU for reimbursement
• Checkoffs "better is enemy of good" - robustness: needs to work in a window
Course Organization (cont)

Checkpoint sequence:

• NEW: CPU -> drive motor+servo -> line sense -> line follow/fig 8 -> velocity control -> high speed steering

Round 1/Round2

Syllabus:
https://inst.eecs.berkeley.edu/~ee192/sp21/docs/syllabus.pdf
Course Organization (cont)

- Emphasis: robustness, simplicity.
  Design/Simulate/Build/Test

- Goal: 10 hours per week per team member.
  - Part of checkpoint: report weekly hours (important for course tuning)

- What about Complexity?

- Reliability of the overall system \((90%)^N\)
  (connectors, power supply, heat sinks, solder joints, CPU stack, car mechanics, camera mount, control stability, lighting robustness,...)
Checkpoint 0

Form a group of 2 (perhaps 3) students. (Remember, choose wisely!)

Have the GitHub usernames of all team members to get a private course GitHub repository.

Note: these repositories will be deleted at the end of the semester for re-use next semester. You are advised to keep a local clone!
CheckPoint 1  Fri Jan.  29

• Install tools, e.g. PlatformIO
• Compile and run SkeletonHuzzah32
• LED fade using PWM
• Timing:
  – How long does a double cos() take?
  – How long does log_add("string"); take compared to snprintf(log,sizeof(log),"string"); log_add(log);
  – How long does it take to print a floating point?
Huzzah32-ESP32

- 240 MHz dual core Tensilica LX6 microcontroller with 600 DMIPS
- Integrated 192 KB instruction SRAM, 200KB Data SRAM, 128KB either
- ESP32-WROOM-32 integrates a 4 MB SPI flash, which is connected to GPIO6, GPIO7, GPIO8, GPIO9, GPIO10 and GPIO11. These six pins cannot be used as regular GPIOs.
- Integrated 802.11b/g/n HT40 Wi-Fi transceiver, baseband, stack and LWIP

- 3 x UARTs
- 3 x SPI, 2 x I2C
- 12 x ADC input channels
- PWM/timer input/output available on every GPIO pin
- OpenOCD debug interface with 32 kB TRAX buffer

Integrated dual mode Bluetooth (classic and BLE)
Ultra-low noise analog amplifier
Hall sensor
10x capacitive touch interface
32 kHz crystal oscillator
2 x I2S Audio
2 x DAC
On-board PCB antenna
SDIO master/slave 50 MHz
SD card interface support
ESP32 Hardware block diagram

Espressif ESP32 Wi-Fi & Bluetooth Microcontroller — Function Block Diagram

Balun
- Switch
- Clock generator
- RF transmit

Radio
- RF receive

Bluetooth
- Bluetooth baseband
- Bluetooth link controller

Wi-Fi
- Wi-Fi baseband
- Wi-Fi MAC

Cryptographic hardware acceleration
- RSA
  - Rivest-Shamir-Adleman
- SHA
  - FIPS PUB 180-4
- RNG
  - Random number gen.
- AES
  - FIPS PUB 197

Core and memory
- Xtensa LX6 microprocessor
  - 32-bit, dual-core or single-core
- ROM
  - Read-only memory
- SRAM
  - Static random-access mem.

RTC and low-power management subsystem
- PMU
  - Power management unit
- Ultra-low-power co-processor
- Recovery memory

Embedded flash memory
- Included in ESP32-PICO-D4 system-in-package QFN module

Peripheral interfaces
- I²C
  - Inter-Integrated Circuit
- SPI
  - Serial Peripheral Interface
- I²S
  - Inter-IC Sound
- SDIO
  - Secure Digital Input Output
- UART
  - Universal async. receiver-transmitter
- CAN
  - Controller Area Network
- ETH
  - Ethernet MAC
- IR
  - Infrared
- PWM
  - Pulse-width modulation
- Temperature sensor
  - Internal, range of -40°C to 125°C
- Touch sensors
  - Ten capacitive-sensing inputs
- DAC
  - Digital-to-analog converter
- SAR ADC
  - Successive approx. analog-to-digital conv.
Software Introduction

• Why Real-time? Bare metal vs Linux
• Measuring Timing
• Tasks in FreeRTOS
• Printing to UART and log task
Software Notes-Basic real time model

Read sensors ➔ process ➔ output ..... Idle ....... Read sensors ➔ process ➔ output

idle

User IO Blocking IO Printf

Interrupt- highest priority

Interrupt- highest priority

Delay leads to instability (for EE128/ME134- negative phase), will show later when discuss control
Example Timing Uncertainty in Linux: main control loop

Calculation loop: input/process/output. Note outliers.
Example Timing Uncertainty in Linux

Using printf in control loop (NOT RECOMMENDED)
Embedded Real-Time Programming with Multiple Tasks

Figure 12.10: Illustration of the priority inheritance protocol. Task 1 has highest priority, task 3 lowest. Task 3 acquires a lock on a shared object, entering a critical section. It gets preempted by task 1, which then tries to acquire the lock and blocks. Task 3 inherits the priority of task 1, preventing preemption by task 2.
FreeRTOS+VS+PlatformIO

• Queue – used across timing domains for coherence of data

• Multiple tasks created at different priority for handling control, sensors, logging, etc
  – Log task: use for non-blocking printf for debugging

• Task List to monitor tasks (time used, stack used)
SkeletonHuzzah32 SW Block Diagram

Keyboard input → user_task → log_queue

control_task → log_queue

timer_evt_task → uart_log_task → UART

wifi_log_task → sendto UDP socket

heartbeat → LED

Note conventions - data flow left to right

main() start tasks and suspend
Measuring Timing from ESP32

tick_start = xTaskGetTickCount();
This gives resolution of 1 ms, depending on CONFIG_FREERTOS_HZ

High resolution timing using built-in 64 bit counter

uint64_t task_counter_value;
double starttime;

timer_get_counter_value(TIMER_GROUP_0, TIMER_0, &task_counter_value);

starttime=(((double)task_counter_value/TIMER_SCALE));
Non-blocking print
https://github.com/ucb-ee192/SkeletonHuzzah32

snprintf(log, sizeof(log),
  "Idle. sum of cos = %d \n", (long)ZSum);
log_add(log); ← snprintf is too slow, can use itoa(), etc

void log_add(char *log)
{
  xQueueSend(log_queue, log, 0);
  // send data to back of queue,
  // non-blocking, wait=0 ==> return immediately if the queue is already full.
}

static void uart_log_task(void *pvParameters)
{
  ...
  xQueueReceive(log_queue, log, portMAX_DELAY);
}
Timing of printf, etc

took about 40 us with log_add(),
but 970 us with snprintf()

Checkpoint 1: measure timing for real-time debugging
Task creation example

/* heartbeat.c*/

static void heartbeatTask(void *pvParameters); //
static=local

if (xTaskCreate(heartbeat, C function "WRITE_TASK_1", configMINIMAL_STACK_SIZE+1024, Stack size for task NULL,
Pointer to parameters to pass into task tskIDLE_PRIORITY + 2, Task priority (0=lowest)
NULL)
Optional handle to created task

!= pdPASS)

{ printf("Task creation failed!\.\r\n\n"));
while(1); // hang indefinitely
}

https://github.com/ucb-ee192/SkeletonHuzzah32
### Monitoring FreeRTOS Tasks - Stack Usage

```c
void print_tasks();
```

# of tasks 12

<table>
<thead>
<tr>
<th>Task name</th>
<th>number of cycles</th>
<th>number of cycles as % of CPU usage</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE0</td>
<td>283440623</td>
<td>49%</td>
<td>0</td>
</tr>
<tr>
<td>usertask</td>
<td>142791054</td>
<td>24%</td>
<td>1</td>
</tr>
<tr>
<td>IDLE1</td>
<td>145004887</td>
<td>25%</td>
<td>1</td>
</tr>
<tr>
<td>heartbeat</td>
<td>6661</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>timer_evt_task</td>
<td>194739</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>Tmr Svc</td>
<td>55</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>control_task</td>
<td>60360</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>esp_timer</td>
<td>209</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>ipc0</td>
<td>10215</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td>95764</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>log_task</td>
<td>13490</td>
<td>&lt;1%</td>
<td></td>
</tr>
<tr>
<td>ipc1</td>
<td>15121</td>
<td>&lt;1%</td>
<td></td>
</tr>
</tbody>
</table>

- Starving the ``idle`` process (will cause a crash).
- Make sure every process has vTaskDelay() for a lower priority process to run
Monitoring FreeRTOS Tasks

```c
void print_tasks();
```

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Priority</th>
<th>Stack</th>
<th>Task#</th>
</tr>
</thead>
<tbody>
<tr>
<td>control_task</td>
<td>R</td>
<td>2</td>
<td>348</td>
<td>14</td>
</tr>
<tr>
<td>usertask</td>
<td>R</td>
<td>0</td>
<td>504</td>
<td>16</td>
</tr>
<tr>
<td>IDLE1</td>
<td>R</td>
<td>0</td>
<td>1116</td>
<td>7</td>
</tr>
<tr>
<td>IDLE0</td>
<td>R</td>
<td>0</td>
<td>1012</td>
<td>6</td>
</tr>
<tr>
<td>heartbeat</td>
<td>B</td>
<td>1</td>
<td>1584</td>
<td>15</td>
</tr>
<tr>
<td>timer_evt_task</td>
<td>B</td>
<td>2</td>
<td>756</td>
<td>13</td>
</tr>
<tr>
<td>Tmr Svc</td>
<td>B</td>
<td>1</td>
<td>1592</td>
<td>8</td>
</tr>
<tr>
<td>main</td>
<td>S</td>
<td>1</td>
<td>2476</td>
<td>5</td>
</tr>
<tr>
<td>log_task</td>
<td>B</td>
<td>1</td>
<td>856</td>
<td>12</td>
</tr>
<tr>
<td>esp_timer</td>
<td>B</td>
<td>22</td>
<td>3640</td>
<td>1</td>
</tr>
<tr>
<td>ipc1</td>
<td>B</td>
<td>24</td>
<td>596</td>
<td>3</td>
</tr>
<tr>
<td>ipc0</td>
<td>B</td>
<td>24</td>
<td>564</td>
<td>2</td>
</tr>
</tbody>
</table>

Priority: 0 is lowest priority. Usertask is also low priority as it busy waits for input
Task #: order of task startup
State: R running, B blocked, S suspend
Class Introductions

Name
Year/major
Location (time zone)
Have a partner/team already?
Extra Slides
Pulse Width Modulation

https://github.com/espressif/esp-idf/tree/b0150615dff529662772a60dcb57d5b559f480e2/examples/peripherals/mcpwm

Also see
~/home/.platformio/packages/framework-espidf/examples/peripherals/mcpwm
Figure 88: LED PWM Output Signal Diagram
LED PWM controller (Ch 15)

Figure 86: LED_PWM High-speed Channel Diagram

Figure 88: LED PWM Output Signal Diagram
Motor Control Pulse Width Modulator (MCPWM) (Ch 17)
Motor Control Pulse Width Modulator (MCPWM) (Ch 17)

Figure 95: Operator Submodule
Motor Control Pulse Width Modulator (MCPWM) (Ch 17)

Figure 95: Operator Submodule

Figure 102: UTEP and UTEZ Generation in Count-Up Mode
- UTEA: the PWM timer is counting up and its value is equal to register A.
- UTEB: the PWM timer is counting up and its value is equal to register B.
- DTEA: the PWM timer is counting down and its value is equal to register A.
- DTEB: the PWM timer is counting down and its value is equal to register B.
A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.
Queue in FreeRTOS

Task A

```c
int x;
x = 20;
```

Task B

```c
int y;
```

Task A changes the value of its local variable before writing it to the queue again. The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).

Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

Figure 31. An example sequence of writes to, and reads from a queue
26. Execution pattern highlighting task prioritization and pre-emption in a hypothetical application in which each task has been assigned a unique priority.
Process ID/Memory Management (Ch 28)

More specifically, when a code tries to access a MMU/MPU-protected memory region or peripheral, the MMU or MPU will receive the PID from the PID generator that is associated with the CPU on which the process is running.

```c
void vTaskAllocateMPURegions(TaskHandle_t xTask, const MemoryRegion_t *const pxRegions)
```
Project Proposal: RTOS timer and threads example

- test_thread1
- test_thread1
- RealTime
- RealTime1
- Main() idle
Instead of Ubuntu: FreeRTOS

The AWS IoT Greengrass Discovery library is used by your microcontroller devices to discover a Greengrass core on your network.

MQTT: lightweight, publish-subscribe network protocol

The AWS IoT Greengrass Discovery library is used by your microcontroller devices to discover a Greengrass core on your network.
Servo PWM

https://www.instructables.com/id/PANTILT-Camera-With-ESP32/