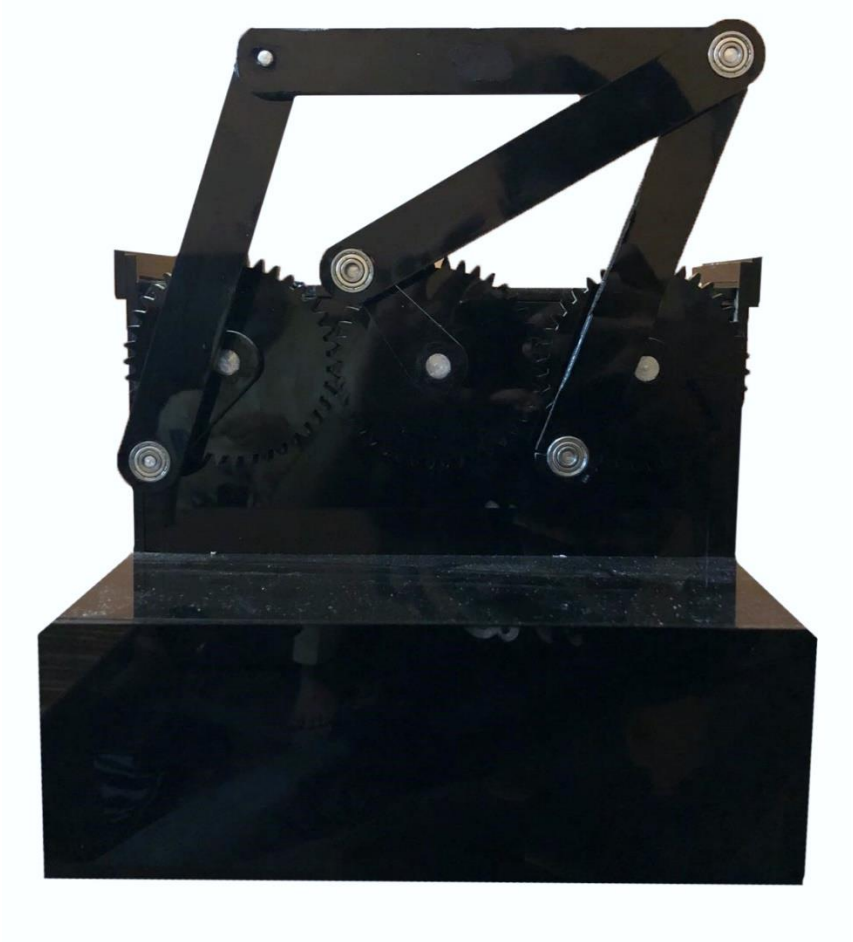
SyncroRun

Arduino Code Documentation



Collaborative Research Project between

NUST and NTUST

Table of Contents

Introduction

This document explains Arduino Code for SyncroRun. The general strategy revolves around frequency and phase matching of trajectories of user and hardware.

Requirements

- Arduino IDE
- Arduino MEGA
- Arduino USB/Serial Converter

Strategy

Instrumentation and Processing

Two sensors have been used: vision sensor for treadmill runner, and IR sensor for hardware mechanism. The computer processes data from vision sensor by detecting and tracking motion of user's face throughout motion, and notifies connected Arduino MEGA through serial pulse whenever a x-maximum point is reached. The IR sensor is configured for corresponding maximum point on hardware. The Arduino constantly listens for serial pulses and IR sensor.

Approach

Based on values from sensors, the Arduino computes 'time ratio' between times taken by user and hardware to complete one cycle. This is similar to the time taken between individual consecutive x-maxima of each.

For frequency matching, whenever values of time taken by user and time taken by hardware to complete a single trajectory are updated, a time ratio is calculated as:

$$Time\ Ratio = \frac{Time\ taken\ by\ Hardware}{Time\ taken\ by\ User}$$

This time ratio is multiplied with Motor PWM in every cycle to manipulate end-effector speed. The motor PWM is bounded with upper and lower limits to prevent speed overshoot and zero speed. Successful frequency matching is represented by Time Ratio equaling 1.

Once frequency is matched, a 'time difference' is calculated between instants the hardware and user hit the x-maxima. If this time difference is within a certain threshold, the phase is considered to be matched. Else, the hardware reaches the x-maxima, and 'waits' until user hits the same maxima. Both then start together.

It is also periodically checked if our changes to motor speed are resulting in synchronization. If the difference in number of trajectories made by each exceeds a certain number, representing synchronization failure, we stop the hardware at maxima, and start it again when user hits the same maxima.

Functions

It is recommended that this section be read in conjunction with underlying code for better understanding.

Global Variables

NAME	TYPE	DESCRIPTION
motorPin	#define	Arduino pin that is used to send PWM to motor driver
sensorPin	#define	Arduino pin that is used to get value from IR sensor
MotorDriverEnablePin	#define	Arduino pin that is fixed at 5V to power motor driver
arduinoTTLModulePowerPin	#define	Arduino pin that is fixed at 5V to power USB/serial converter
maximaUser	int	Stores 1 or 0 for status of user hitting x-maxima
maximaHardware	int	Stores 1 or 0 for status of hardware hitting x-maxima
userMaximaCount	int	Count of trajectories made by user
hardwareMaximaCount	int	Count of trajectories made by hardware
userMaximaTime	double	Timer value when user hits maxima
hardwareMaximaTime	double	Timer value when hardware hits maxima
userMaximaTime_prev	double	Previous timer value when user hits maxima
hardwareMaximaTime_prev	double	Previous timer value when hardware hits maxima
motorPwm	double	PWM being sent to the motor driver
motorPwm_prev	double	For storing previous motor PWM value
time_diff	double	Time difference between user and hardware hitting same maxima
time_ratio	double	Time ratio between trajectory times of hardware and user
time_ratio_prev	double	To store previous time ratio value
maximaHardwareSensorTime	int	For delay in IR value retrieval to ensure maximaHardware gets set to 1 only once in a single trajectory
changeMotorPwm	bool	To ensure motor PWM gets modified only once during frequency matching
discardFirstSerialVal	bool	To discard first user maxima serial pulse caused by any present serial garbage data
motorPwm_lowerLimit	int	Lower limit for Motor PWM
motorPwm_upperLimit	int	Upper limit for Motor PWM
syncedPhase	bool	Identifies whether phase matching has finished

void setup()

Runs once after the Arduino is reset. Configures serial baud rates: 9600 for communication with Qt GUI for serial pulse of user maxima, and 115200 for writing output data to Arduino Serial Monitor, for debugging. Also sets status of output pins.

void loop()

Runs code inside it in an infinite while loop. Used to run functions one by one.

void ResetVariables()

Resets values of all global variables.

void Step1()

Used to drive hardware to x-maxima, and stop it. Constantly reads sensor value. If maximum arrives, stops motor and proceeds to next function. Else, keeps running motor with initial speed.

void Step2()

Constantly listen for user maxima serial pulse. Discard first pulse as garbage. At second pulse, note microcontroller timer value, allocating it to variables of both user and hardware maxima timers. This is because both are at the same point.

void Step3()

This function is where Arduino constantly attempts to match frequency and phase of user and hardware.

Constantly

- Listen for both IR sensor values and user maxima serial pulses.
- Check if user maxima serial pulse is received, note current timer value and store previous timer value. Also increment number of user maxima counts.
- Check if IR sensor value denoting maxima is received, note current timer value and store previous timer value. Also increment number of hardware maxima counts.
- Compute time difference. If frequency matched (equal count of user and hardware maxima), and time difference greater than a threshold (phase not matched), proceed to Step5()
- Check if count of user maxima and hardware maxima is equal, to compute a reliable ratio. If yes, compute time ratio and modify motor PWM for frequency matching.
- Check if difference between counts of user trajectories and hardware trajectories has exceeded 2. If yes (meaning that mechanism is totally out of sync), proceed to Step4().

void Step4()

This function stores current motor PWM in a separate variable, and stops mechanism at x-maximum, waiting for corresponding user maximum signal. When the signal arrives, it recalculates time ratio, checking if it's greater than a certain threshold. It then multiplies the previously stored PWM with new time ratio, and ensures that resulting motor PWM is within set limits, before resetting maxima counts. The result is motor running at standard PWM, modified by latest time ratio, resulting in frequency and phase matching owing to user and hardware starting from the same point.

void Step5()

This function is used for phase matching. It is same as Step4(), except that it does not recalculate time ratio. It stops hardware at x-maximum, and waits for user's maximum signal. Once reached, it jump starts motor with a fixed PWM, restoring old PWM soon after, letting Step3() do remaining calculations.

void processSensorValue()

This sensor is meant to be run in a loop, enabling Arduino to reliably read value from IR sensor. The indicator used on gear to identify mechanism maximum could be of variable width, resulting in IR sensor reading false positives throughout indicator's traversal, for a single cycle. To restrict that and reliably read single maximum value for each cycle, this function sets maximaHardware variable value high the first time, and then discards future high values for a certain time. The delay is invoked by storing timer value the first time, and computing difference with new timer values to check elapsed time.

void printValues()

Prints values of various global variables to Arduino Serial Monitor output screen for debugging.

void getSerial()

Used to be run in a loop. Checks if a serial pulse has been sent for user's trajectory maximum. If yes, raises the flag maximaUser to high.

Conclusion

Synchronization was achieved successfully at different user speeds. The mean time ratio achieved for various users ranged from 0.95 to 1.05. Phase matching was done as soon as time difference became large. Work can be done to make experimental setup simple enough for a user to simply mount the treadmill and start running, instead of having to reset Arduino and ensure successfully detected face pre-running.

Video Demo

Video demos can be found here:

1. Concept Animation: https://youtu.be/csyDfKPXqz8
2. Demo 1: https://youtu.be/MMvzMPrRD8o
3. Demo 2: https://youtu.be/-5uN_vG9Avs

It is recommended that Captions/Subtitles be turned on while watching Demo videos.

Code

Code for the project be found at https://github.com/abdullahgulraiz/NTUST-Treadmill-Project.

GUI Requirements: OpenCV 3.3, Qt. Tested on Ubuntu 16.04 LTS.