TiSCo project

A framework of robot control software and IT hardware to complement the mechanical side of generic educational and hobby robots.

About The Project

The project provides a simple, uniform way to set all kinds of complex mechanisms that can be built around 16 hobby servos in motion.

Programming a Robot by Teaching it!

A key property (lacking in most hobby robot control software) is *Teach-in* functionality: just manipulate the robot joints *by hand* in real time, 'showing' it how to move while the joint angles are recorded. After processing the input, the robot fluently replays the recorded motions — at any desired speed. This enables extensive and complex motions without ever applying any complex robotics theory like Inverse Kinematics.

Why this project?

The project allows anyone that wants to get their hands dirty and *build* things to bring their projects to life. It frees them to focus on the *mechanical* side of their designs: material selection, mechanical strength, stiffness-to-weight ratio, mass/weight, durability, manufacturability, sustainability, cost ...

After the mechanical design is validated, the mechanism can be passed on to software and microcontroller enthusiasts for developing proper robotics control firmware.

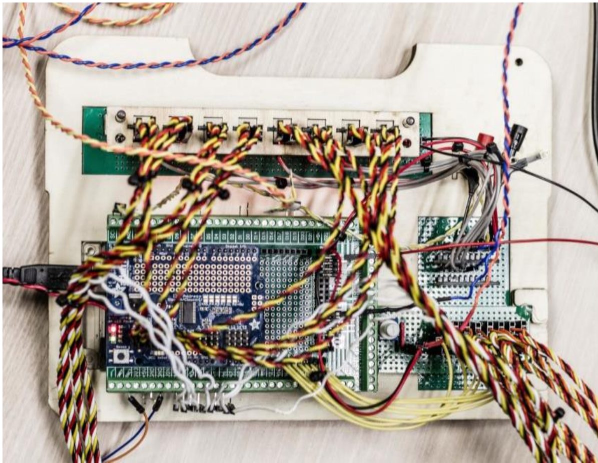
Key Features

- Based on a standard Arduino Mega 2560 with extra shields and breakout boards, and standard hobby servos with small modification.
- Controls up to 16 modified hobby servos (our prototype is limited to 8 servos).
- No specialized robotics theory like Inverse Kinematics is implemented. Complex motions are simply taught using the *Teach-in* feature: just manipulate the joints by hand while the joint angles are recorded.
- Three modes of motion: directly mirroring joystick input (normal joystick or custom
  - made using low cost potentiometers),
  - replaying an earlier recorded motion, executing a stream of COSMOS
  - commands, setting the desired joint angles.
- Active servo protection by cutting of the servos power supply after current overload is detected (to be implemented).

This repository deals exclusively with the electronics and software. For the mechanical side of the robots, see the LORE Robot project homepage!

(see also what the project doesn't do)

Hardware



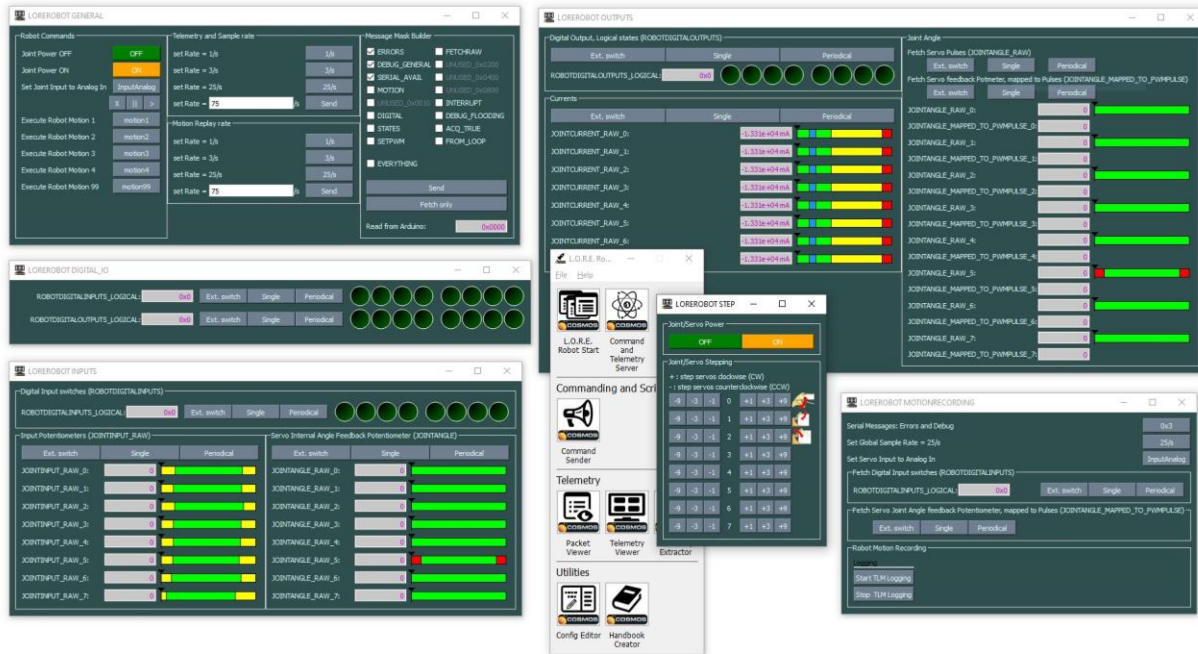
The controller is a standard Arduino Mega 2560 rev. 3 topped with a 16 channel PWM servo shield, driving up to 16 modified hobby servos (Hitec HS-625MG and HS-645MG, eight are connected in the prototype of which three actually drive the mechanism). Power to the servos is fed by one or more *Weidmuller CP E SNT 100W 5V, 16A* Switched Mode Power Supply.

A few ADS1015 analog to digital breakout boards together with lots of analog input pins, register the angle of modified low cost potentiometers as well as the servos internal shaft angle.

Small ACS712 current sensor boards monitor each servos current draw. When a servo should be overloaded, one or all servos are automatically deactivated to prevent further damage (yet to be implemented).

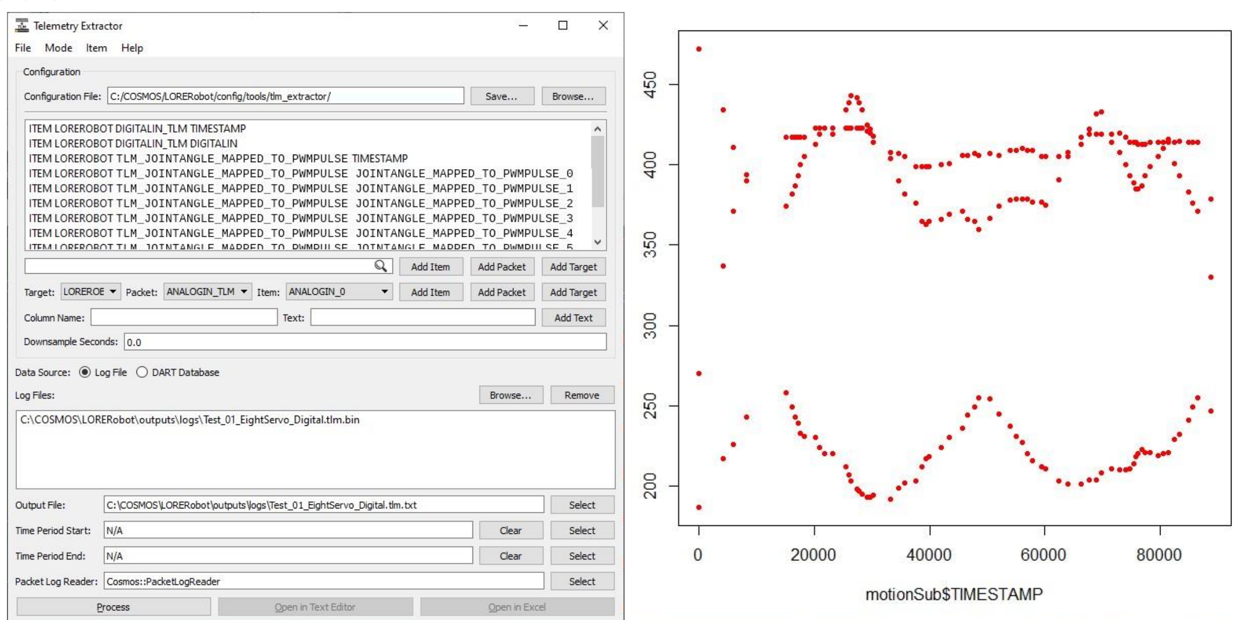
(back to top)

COSMOS/OpenC3 Control Software



The robot is currently controlled using Ball Aerospace COSMOS ("The User Interface for Command and Control of Embedded Systems"). Using Command and Telemetry 'screens', all operational instructions, detailed telemetry and test commands are passed via serial protocol over USB cable. Soon, the control software will be updated to version 5.x of COSMOS' successor, OpenC3.

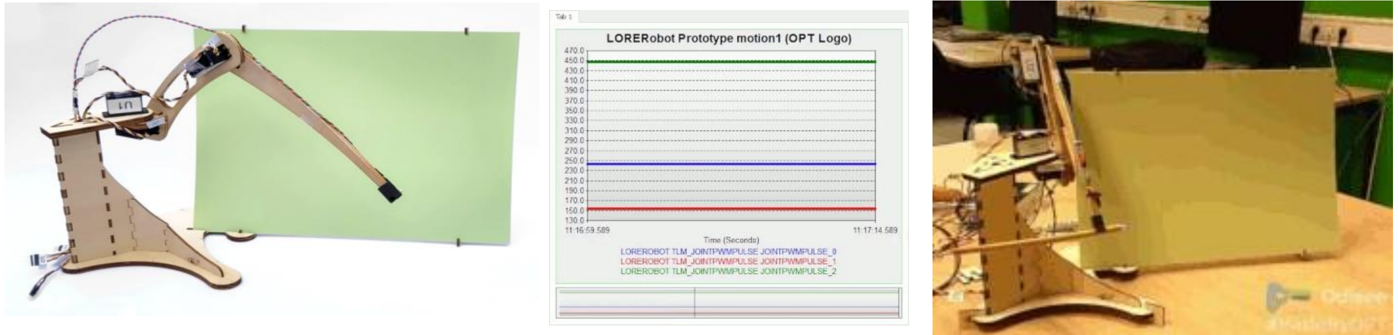
Check out COSMOS Commands and Telemetry handbook for this project, for all communication details.



COSMOS allows for all telemetry data to be stored and to be extracted for further analysis. An R script cleans up, curve-fits and resamples the raw telemetry data. This results in an C++ include file called `motion1.h` (1 or higher) that contains the motion data, ready to be (re)compiled in the firmware.

(back to top)

'Proof of Concept' Development Prototype



Using the *Teach-In* functionality, it's a breeze to set *any* mechanical construction in motion. For example, the image shows a three DOF 'proof-of-concept' made from 4mm plywood. It lacks the stiffness for proper functionality, obviously; it is used for designing and testing the firmware and COSMOS control software.

See it in action while it draws a logo on a phosphorescent screen with an UV LED. The motion was recorded earlier by simply tracing the logo from a piece of paper atop the screen (video from LORE Youtube playlist). Using a Telemetry screen, the used joint PWM pulses can be retrieved and displayed while the motion is running.

What the Project Doesn't Do

- The robots don't move autonomously, following high level commands like 'walk forward' or 'grip object'. Either they mirror analog inputs in real time, they replay earlier recorded motions or they set the joints according to an incoming stream of COSMOS command packets.
- There is no support for rotary motions like wheels. This means that Turtle robots or other mobile robots are not feasable yet. If a reliable teach-in method for multiple rotations can be devised, this might be included.
- Battery power supply is no priority (since the robots can't move around very far) and so is power consumption. The robot remains tethered to its power supply.

Roadmap

- [ ] Write detailed instructions on:
  - [ ] Calibration of input potentiometers and joysticks
  - [ ] Calibration of servo internal angle feedback potentiometers, setting the safe software limits for servo sweep angles.
  - [ ] Calibration of current sensors
  - [ ] COSMOS usage
  - [ ] recording and replaying motions
- [ ] Improve code comments, expanding the Doxygen generated documentaion.
- [ ] Upgrade the prototype mechanism for much higher stiffness using the Markforged Mark Two Continuous Fiber Composite 3D Printer, and create comprehensive tutorials and demos
- [ ] Create Battle of the Lores video, Round 2 and 3!

Contributing

Since this project will be featured in an educational setting, it is important that it serves as an example of good code practices and use of embedded Object Oriented Programming using C++. Also, the coding style should preferrably not differ too much from established Arduino coding styles.

Prioritized todos

Coding style

Evaluate the code and suggest better (embedded) coding practices:

- What kind of classes go into which file(s)? How to organize class definitions in well-chosen files?
- Correct usage of include files (.h) vs. code files (.cpp).
- Proper implementation of Object Oriented Programming with abstraction, encapsulation and inheritance (for code reuse), keeping in mind that the code is running on embedded hardware.
  - inheritance <-> composition
  - public/private members and variables
- Make code more readable and maintainable.
- If sensible, refactor large switch() statements using polymorphism, for readability.

Feature requests

- Implementation of safe serial framed data communication, possibly using COBS or SLIP protocols. Currently, their seems no standard way to implement COBS or SLIP in COSMOS interfaces (a custom protocol must be written in Ruby).
- Move initialization of all joint and robot parameters that are now hardcoded in calibration_and_settings.h to COSMOS. The robot should be initialized after booting, via a series of COSMOS initialization commands.
- Upload/download/verify robot motions via COSMOS instead of compiling the motions together with the code.
- Move all debug message strings to flash memory instead of SRAM.
- Write test procedures for testing the code base and COSMOS communication.
- Upgrade COSMOS 4.x to latest 5.x implementation.

If you want to help out, please fork the repo and create a pull request.

Don't forget to give the project a star! Thanks again!

License

Distributed under the GPL-3.0 License. See `./LICENSES` for more information.

Languages