FORDYCA Documentation

Release 2.35.0.0

John Harwell

Apr 23, 2023

CONTENTS:

CHAPTER ONE

SETUP

1.1 Building The Code

Head over to https://github.com/swarm-robotics/bootstrap.git to download and build FORDYCA and all of its dependencies.

1.1.1 Local Runtime Setup

If you have not successfully completed the build part of the setup, do that first. These steps will not work otherwise.

After successful compilation, follow these steps to setup the FORDYCA runtime environment and run a basic foraging scenario on your local laptop.

Note: If you don't want to go through this runtime setup each time you start a new shell, add whatever commands you run in the terminal to `$HOME/.bashrc` (or whatever the startup file for your shell is) to have them run automatically when you login.

1. Update the *system* dynamic library search paths so the OS can find the libraries that the ARGoS executable requires (supposedly ARGoS will do this for you when you install it via ldconfig to `/usr/local`, but many people still have trouble with it). On bash:

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/.local/system/lib/argos3:$HOME/.local/
   ↪lib/argos3
   ```

   Assuming you passed `--sysprefix=$HOME/.local/system` and `--rprefix=$HOME/.local` to `bootstrap.sh`. If you passed something else, then update the path above accordingly.

2. Update your `PATH` so that the shell can find ARGoS. On bash:

   ```
   export PATH=$PATH:/opt/.local/system/bin
   ```

   Assuming that you passed `--sysprefix=$HOME/.local/systembin` to `bootstrap.sh`. If you passed something else, then update the above path accordingly.

3. Set the `ARGOS_PLUGIN_PATH` variable to contain (1) the path to the directory containing the `libfordyca.so` file, (2) the path to the ARGoS libraries. On bash, that is:

   ```
   export ARGOS_PLUGIN_PATH=/$HOME/.local/system/lib/argos3/lib:$HOME/research/fordyca/
   ↪build/lib
   ```

Assuming that you passed `--sysprefix=$HOME/.local/system --rroot=$HOME/research` to `bootstrap.sh` script when you built FORDYCA. If your paths are different, modify the above paths accordingly. Note that you need BOTH of these terms in the path, because this defines the ENTIRE search space for argos to look for libraries (including its own core libraries).

4. Unless you compile out event reporting (built FORDYCA with optimizations *AND* with `LIBRA_ER=NONE` passed to cmake), you will need to set the path to the log4cxx configuration file, which tells FORDYCA which classes should have logging turned on, and how verbose to be. On bash that is:

   ```
   export LOG4CXX_CONFIGURATION=$HOME/research/fordyca/log4cxx.xml
   ```

   Assuming you have cloned and built FORDYCA in `$HOME/research`. If you cloned and built it somewhere else, then update the above path accordingly.

5. cd to the ROOT of the FORDYCA repo, and run the demo experiment:

   ```
   argos3 -c $HOME/research/fordyca/exp/demo.argos
   ```

   This should pop up a nice GUI from which you can start the experiment (it runs depth0 dpo foraging by default). If the simulation seems to start but no GUI appears, verify that the `<visualization>` subtree of `demo.argos` file is not commented out.

Important: You should (probably) have `n_threads` set to 0 in the `.argos` file or omit the attribute altogether when running debug builds. When debugging you want things to be executed in a deterministic manner, and non-deterministic parallel execution with multiple threads should be used only for optimized builds once you are confident of code correctness.

Runtime Issues

Before reporting a bug, try:

1. If you are getting a segfault when running ARGoS, verify that if you are running with Qt visualizations that the threadcount is 0 (Qt5 cannot run with multiple threads without segfaulting).
2. Verify you don't have any anaconda bits in your `PATH`. Depending on version, anaconda loads a DIFFERENT version of the Qt than fordyca uses, resulting in a dynamic linking error.
3. Make sure you have the necessary environment variables set correctly.
4. If you get a `std::bad_cast`, `boost::get`, or similar exception, then verify that the name of [controller, loop functions, qt user functions], match, as specified in *XML Configuration* are correct.

1.1.2 MSI Setup

Head over to SIERRA, and follow the MSI setup instructions over there. Don't try to run on MSI without SIERRA. Just don't.

CHAPTER

TWO

XML CONFIGURATION

2.1 Controller XML Configuration

The following controllers are available:

Controller	Required Loop Functions	Notes
crw	d0	CRW = Correlated Random Walk.
dpo	d0	DPO = Mapped Decaying Pheromone Object. Uses pheromones to track objects within the arena.
mdpo	d0	MDPO = Mapped Decaying Pheromone Object. DPO + mapped extent of the arena tracking relevance of individual cells within it.
odpo	d0	ODPO = Oracular DPO. Has perfect information about blocks in thye arena.
omdpo	d0	OMDPO = Oracular MDPO. Has perfect information about blocks in the arena.
bitd_dpo	d1	Greedy task partitioning + DPO. Requires static caches to also be enabled.
bitd_odpo	d1	Greedy task partitioning + DPO + oracle (perfect knowledge, as configured). Requires static caches, oracle to be enabled.
bitd_mdpo	d1	Greedy task partitioning + MDPO. Requires static caches, oracle to be enabled.
bitd_omdpo	d1	Greedy task partitioning + MDPO + oracle (perfect knowledge, as configured). Requires static caches, oracle to be enabled.
birtd_dpo	d2	Recursive greedy task partitioning + DPO. Requires dynamic caches to be enabled.
birtd_mdpo	d2	Recursive greedy task partitioning + MDPO. Requires dynamic caches to be enabled.
birtd_odpo	d2	Recursive greedy task partitioning + DPO + oracle (perfect knowledge, as configured). Requires dynamic caches, oracle to be enabled.
birtd_omdpo	d2	Recursive greedy task partitioning + MDPO + oracle (perfect knowledge, as configured). Requires dynamic caches, oracle to be enabled.

The following root XML tags are defined under <params>.

Root XML Tag	Mantory For ?	Notes
perception	All but CRW	See COSM docs also; only augmented slightly here.
block_sel_matrix	All but CRW	Parameters used by robots when selecting which block to acquire as part of the task they are currently executing.
cache_sel_matrix	All d1, d2 controllers	Parameters used by robots when selecting which cache to acquire as part of the task they are currently executing.
sensing_subsystemQ3D	All controllers	See COSM docs.
actuation_subsystem2D	All controllers	See COSM docs.
strategy	All controllers	Parameters for robot exploration, collision avoidance, etc. strategies.
task_executive	d1, d2 controllers	See COSM docs.
task_alloc	d1, d2 controllers	See COSM docs.

2.1.1 block_sel_matrix

- Required child attributes if present: nest.
- Required child tags if present: none.
- Optional child attributes: none.
- Optional child tags: [block_priorities, pickup_policy].

XML configuration:

```
<block_sel_matrix
    nest="6, 3">
    <block_priorities>
    ...
    </block_priorities>
    <pickup_policy>
    ...
    </pickup_policy>
</block_sel_matrix>
```

nest - The location of the nest.

block_sel_matrix/block_priorities

- Required by: None. If omitted, the default priority values shown below are used.
- Required child attributes if present: nest.
- Required child tags if present: none.
- Optional child attributes: [cube, ramp].
- Optional child tags: none.

XML configuration:

```
<block_sel_matrix>
    ...
    <block_priorities
    cube="1.0"
    ramp=1.0/>
    ...
</block_sel_matrix>
```

- `cube` - The priority value used as part of block utility calculation for cube blocks during block selection. Default if omitted: 1.0
- `ramp` - The priority value used as part of block utility calculation for ramp blocks during block selection. Default if omitted: 1.0

`block_sel_matrix/pickup_policy`

- Required by: None.
- Required child attributes if present: `policy`.
- Required child tags if present: none.
- Optional child attributes: [`cluster_proximity`, `prox_dist`].
- Optional child tags: none.

XML configuration:

```
<block_sel_matrix>
    ...
    <pickup_policy
    policy=""
    prox_dist="FLOAT"/>
    ...
</block_sel_matrix>
```

- `policy` - The policy to use to restrict (1) the conditions under which robots can pick up a block that they encounter, (2) which blocks are considered valid for acquisition. Valid values are:
  - `cluster_proximity` - Only allow blocks which are within `prox_dist` from the average of the positions of the blocks currently known to a robot to be picked up. Only makes sense for object distributions in which objects are clumped into clusters; used to help d2 robots not to pickup the blocks other robots have dropped in order to start caches.
  - `""` - An empty string to disable if the the tag `pickup_policy` is present.
- `prox_dist` - The minimum distance measure for usage with `cluster_proximity` pickup policy.

2.1.2 cache_sel_matrix

- Required by: [d1, d2] controllers.
- Required child attributes if present: all.
- Required child tags if present: none.
- Optional child attributes: none.
- Optional child tags: `pickup_policy`.

XML configuration:

```
<cache_sel_matrix
    cache_prox_dist="FLOAT"
    nest_prox_dist="FLOAT"
    block_prox_dist="FLOAT"
    site_xrange_dist="FLOAT:FLOAT"
    cache_prox_dist="FLOAT:FLOAT">
        <pickup_policy>
        ...
        </pickup_policy>
</cache_sel_matrix>
```

- `cache_prox_dist` - When executing the Cache Finisher task, the constraint applied to new cache selection for how close the chosen new cache can be to known existing caches. Should be at least twice the size of a cache for Cache Finisher robots to behave properly and not get stuck in infinite loops attempting to drop a block too close to a known cache.
- `block_prox_dist` - When executing the Cache Starter task, the constraint applied to cache site selection for how close the chosen cache site can be to known blocks.
- `nest_prox_dist` - When executing the Cache Starter task, the constraint applied to cache site selection for how close the chosen cache site can be to the nest.
- `site_xrange` - The valid X range for cache site selection (should be a subset of the full arena X size, to avoid robots being able to select locations by arena boundaries).
- `site_yrange` - The valid Y range for cache site selection (should be a subset of the full arena Y size, to avoid robots being able to select locations by arena boundaries).

cache_sel_matrix/pickup_policy

- Required by: [d1, d2] controllers.
- Required child attributes if present: `policy`.
- Required child tags if present: none.
- Optional child attributes: [`timestep`, `cache_size`].
- Optional child tags: none.

XML configuration:

```
<cache_sel_matrix>
    ...
    <pickup_policy
        policy="time|cache_size|cache_duration"
```

(continues on next page)

(continued from previous page)

```
        timestep="INTEGER"
        cache_size="INTEGER"/>
    ...
</cache_sel_matrix>
```

- policy - The policy to use to restrict (1) the conditions under which robots can pick up from a cache that they encounter, (2) which caches are considered valid for acquisition. Valid values are:
  - cache_size - Only allow robots to pickup from caches with at least cache_size blocks in them. Robots intending to drop blocks in caches are not restricted.
  - cache_duration - Only allow robots to pickup from caches after they have existed for at least time timesteps.
  - time - Only allow robots to pickup from caches after timestep timesteps have elapsed during simulation. Robots intending to drop blocks in caches are not restricted.
  - Can also be an empty string to disable the cache pickup policy if the pickup_policy tag is present.

2.1.3 strategy

- Required by: All controllers.
- Required child attributes if present: None.
- Required child tags if present: [explore, nest, blocks].
- Optional child attributes: [caches]
- Optional child tags: None.

XML configuration:

```
<strategy>
    <blocks>
      ...
    </blocks>
    <caches>
      ...
    </caches>
    <nest>
      ...
    </nest>
</strategy>
```

perception

- Required child attributes if present: [type].
- Required child tags if present: none.
- Optional child tags: [rlos, dpo, mdpo]
- Optional child attributes: none.

XML configuration:

```
<perception
  type="STRING">
  <rlos>
     ...
  </rlos>
  <dpo>
     ...
  <dpo/>
  <mdpo>
     ...
  <mdpo/>
</perception>
```

- `type` - The perception type to use.

`perception/dpo`

Parameters for the Decaying Pheromone Object (DPO) perception type.

- Required child attributes if present: none.
- Required child tags if present: [`rlos`, `pheromone`].
- Optional child tags: none.
- Optional child attributes: none.

XML configuration:

```
<perception>
  ...
  <dpo>
    <rlos>
      ...
    </rlos>
    <pheromone>
      ...
    </pheromone
  </dpo>
  ...
</perception>
```

`perception/dpo/pheromone`

Parameters controlling the decay of the pheromone-based memory for the Decaying Pheromone Object (DPO) perception model.

- Required child attributes if present: `rho`.
- Required child tags if present: none.
- Optional child attributes: `repeat_deposit`.
- Optional child tags: none.

XML configuration:

```
<dpo>
  ...
  <pheromone rho="FLOAT"
             repeat_deposit="false"/>
  ...
</dpo>
```

- `rho` How fast the relevance of information about a particular cell within a robot's 2D map of the world loses relevance. Should be < 1.0.
- `repeat_deposit` - If *true*, then repeated pheromone deposits for objects a robot already knows about will be enabled. `rho` should be updated accordingly, probably to a larger value to enable faster decay. Default if omitted: *false*.

`perception/mdpo`

Parameters for the Mapped Decaying Pheromone Object (MDPO) perception model.

- Required child attributes if present: none.
- Required child tags if present: [`rlos`, `pheromone`].
- Optional child tags: none.
- Optional child attributes: none.

XML configuration:

```
<perception>
  ...
  <mdpo>
    <rlos>
      ...
    </rlos>
    <pheromone>
      ...
    </pheromone
  </mdpo>
  ...
</perception>
```

2.1.4 Additional notes to COSM controller docs

`task_alloc/stoch_nbhd1`

- `tab_sel` child tag required by d2 controllers

task_alloc/task_exec_estimates

Valid values for <task_name> are:

- generalist
- collector
- harvester
- cache_starter
- cache_finisher
- cache_transferer
- cache_collector

2.2 Loop Functions XML Configuration

The following root XML tags are defined under <loop_functions>:

Root XML Tag	Mandatory For?	Description
output	All	See COSM docs.
convergence	None	See COSM docs.
arena_map	All	See COSM docs.
temporal_variance	None	See COSM docs.
visualization	None	See COSM docs.
oracle_manager	None	See COSM docs.
caches	Depth1, depth2 controllers	Parameters for the use of caches in the arena.

Any of the following attributes can be added under the metrics tag in place of one of the <append>,<create>, <truncate> tags, in addition to the ones specified in COSM. Not defining them disables metric collection of the given type.

Extend the temporal variance capabilities in COSM with caches:

2.2.1 temporal_variance/env_dynamics/caches

- Required by: none.
- Required child attributes if present: none.
- Required child tags if present: none.
- Optional child attributes: none.
- Optional child tags: [usage_penalty].

XML configuration:

```
<temporal_variance>
    ...
    <caches>
        <usage_penalty>
        ...
```

(continues on next page)

(continued from previous page)

```
        </usage_penalty>
    </caches>
    ...
</temporal_variance>
```

2.2.2 temporal_variance/caches/usage_penalty

- Required by: none.
- Required child attributes if present: none.
- Required child tags if present: `waveform`.
- Optional child attributes: none.
- Optional child tags: none.

XML configuration:

```
<caches>
    ...
    <usage_penalty>
    <waveform>
        ...
    </waveform>
    </usage_penalty>
    ...
</caches>
```

- `waveform` - Parameters defining the waveform of cache usage penalty (picking up/dropping).

Extend the arena map capabilities in COSM with caches:

arena_map/caches

- Required by: [depth1, depth2 controllers].
- Required child attributes if present: [`dimension`, `strict_constraints`].
- Required child tags if present: none.
- Optional child attributes: none.
- Optional child tags: [`static`, `dynamic`].

XML configuration:

```
<arena_map>
    ...
    <caches
        dimension="FLOAT"
        strict_constraints="true">
        <static>
            ...
        </static>
        <dynamic>
```

(continues on next page)

(continued from previous page)

```
            ...
        </dynamic>
    </caches>
    ...
</arena_map>
```

- `dimension` - The dimension of the cache. Should be greater than the dimension for blocks.
- `strict_constraints` - If *true*, then created caches will not be checked for overlap with block clusters in the arena after creation (this happens in non-error contexts with static caches and RN block distributions, for example). Other sanity checks will still be performed and appropriate error messages issued; however, an "OK" return code will always be returned.

  For dynamic cache creation, if *true*, cache creation will be strict, meaning that any caches that fail validation after creation will be discarded. This can happen because when robots select cache sites they only consider the distance between the *center* of existing caches/blocks/nests/etc, and do not take the extent into consideration. Depending on what the values of the various proximity constraints robots use when searching for a cache site, validation can fail after cache creation.

  For dynamic cache creation, if *false*, then dynamically created caches will be kept regardless if they violate constraints or not, which MIGHT be OK, or MIGHT cause issues/segfaults. Provided as an option so that it will be possible to more precisely duplicate the results of papers run with earlier versions of FORDYCA which had more bugs.

  For static cache creation, caches are never discarded; however if one or more caches fail validation after creation, an assert will be triggered if set to *true*.

  Default if omitted: *true*.

2.2.3 `arena_map/caches/static`

- Required by: [depth1 controllers].
- Required child attributes if present: [`enable`].
- Required child tags if present: none.
- Optional child attributes: [`size`, `respawn_scale_factor`].
- Optional child tags: none.

XML configuration:

```
<caches>
    ...
    <static
        enable="false"
        size="INTEGER"
        respawn_scale_factor="FLOAT"/>
    ...
</caches>
```

This tag is required for `depth1` loop functions. If the tag is present, only the `enable` attribute is required; all other attributes are parsed iff `enable` is *true*.

- `enable` - If true, then a single static cache will be created in the center of the arena. The cache will be replenished by the loop functions if robots deplete it, under certain conditions.

- `size` - The number of blocks to use when (re)-creating the static cache. Must be >= 2.
- `respawn_scale_factor` - A scale factor controlling how quickly the probability of static cache respawn will grow once the conditions for respawning are met.

2.2.4 `arena_map/caches/dynamic`

- Required by: [depth2 controllers].
- Required child attributes if present: `enable`.
- Required child tags if present: none.
- Optional child attributes: [`min_dist`, `min_blocks`, `robot_drop_only`].
- Optional child tags: none.

XML configuration:

```
<caches>
    ...
    <dynamic
        enable="false"
        min_dist="FLOAT"
        min_blocks="INTEGER"
        robot_drop_only="false" />
    ...
</caches>
```

- `enable` - If *true*, then the creation of dynamic caches will be enabled.
- `min_dist` - The minimum distance between blocks to be considered for cache creation from said blocks.
- `min_blocks` - The minimum # of blocks that need to within `min_dist` from each other to trigger dynamic cache creation.
- `robot_drop_only` - If *true*, then caches will only be created by intentional robot block drops rather than drops due to abort/block distribution after collection. Default if omitted: *false*.

2.3 XML Conventions

- Multiple choices for an XML attribute value are separated by a `|` in the example XML.
- XML attributes that should be floating point are specified as `FLOAT` in the example XML (acceptable range, if applicable, is documented for each individual attribute).

CHAPTER

THREE

CONTRIBUTING

3.1 Parser Tutorial

After you have added some new code to FORDYCA, you will (probably) need to be able to configure your new module(s) from the input `.argos` file. To do that you need to define a new *XML parser*.

This is a tutorial for:

- Adding parameters to the input file
- Adding a parameter struct to the code for said input parameters
- Defining a parser for said input parameters
- Registering said parser so that it is called during initialization.

It assumes that you have already built the documentation for forydca and rcppsw and have some level of familiarity with the XML parameter parsing.

3.1.1 Adding Parameters To Input File

1. Identify what parameters you want to add (i.e. what new knobs you want to be able to fiddle with).
2. Decide what type of parameter each of the new ones you want to add is: robot or simulation. Each type has a different section of the input file that they have to go into.

   Robot parameters are things that robots need to do whatever you've told them to do. Simulation parameters are for things that are not specific to a single robot. Robot parameters are found under the robot controller section, and simulation parameters under the loop function section (see `exp/testing.argos` for good examples of existing parameters).

   All your new parameters may be only in one category, and that's fine.
3. Find an appopriate place in the input file to place said parameters. You should add at most 1 XML tag to the robot/simulation parameter section of the input file (i.e. all your new parameters for each category should be able to be grouped logically/hierarcically under a single XML tag). If you are not sure where is appropriate, ask.

   Pick a GOOD name for the root XML tag you add, as that is very important for the next step.

   For example, you might have:

   ```
   <widget>
      <subwidget
         param1="10"
         param2="FOOBAR"/>
   </widget>
   ```

3.1.2 Adding _config Struct

At an appropriate location in the fordyca::config hierarchy, create a new configuration struct in a .hpp file. There are literally dozens of examples to look at already present in the code, many of which are very simple.

- The parameter struct should be named <XML tag>_config to help with readability and the principle of least surprise. For example, if your XML tag name is subwidget, then your parameter struct would be subwidget_config. This is the Principle of Least Surprise at work.
- Each element of the parameter struct should have the SAME name as one of the XML attributes under the root tag in the input file, to help with readability and the principle of least surprise. For example, if you have an attribute called rate under <energy_consumption, then in energy_consumption_config you would also have a member called rate, of whatever type is needed. This is enforced during parsing in the C++ code.

  For our <subwidget> example, we would defined:

  ```
  struct subwidget_config {
    int param1;
    std::string param2;
  }
  ```

3.1.3 Defining an XML Parser

At an appropriate location in the fordyca::config hierarchy, create a new parser in .hpp/.cpp files. There are literally dozens of examples to look at already present, many of which are very simple. The parser should be named <XML tag>_parser to help with readability and the Principle of Least Surprise. For example, if your tag name is subwidget, then your parser name would be subwidget_parser.

When creating your class, you *must* define the following inside the public access modifier:

- config_type - Set to the type of the config struct for your class. This is a convention/requirement that allows lookup of the specific type of config a parser is parsing without casing on the parser name. Because of the above conventions, this should be the name of your parser class, changing _parser to _config.

When creating your class, you *must* override:

- *config_get_impl()* - This returns a non-owning reference to the internal parsed config struct. If the config struct has not been populated (e.g., the necessary tag for the parser to parse was not in the input .argos file), then it should return nullptr.
- xml_root() - Return the name of the XML tree that your parser is parsing. Because of the conventions used, this should be the name of your parser class, minus the _parser at the end.
- parse(): Does the actual parsing. The function is passed the ticpp::Element node of the *parent* XML tag that your parser is parsing; this convention allows easy parser nesting and clean parsing regardless if an XML tag is expected to exist or not.

  For example, if you are defining a subwidget_parser class, you might have the following XML structure:

  ```
  <widget>
     <subwidget
        param1="10"
        param2="FOOBAR"/>
  </widget>
  ```

  With such an XML structure, your subwidget_parser::parser() function will be passed a reference to the <widget> tree, and you will need to call node_get("subwidget") to get a reference to the XML tree rooted at <subwidget>.

You MUST use the XML_PARSE_ATTR() macro to do most parsing, so that if you do not name struct members with the same name as the input file attribute, you will get compile time rather than run time errors. If you want to have an optional attribute, you can supply a default via XML_PARSE_ATTR_DFLT(). Otherwise, a missing attribute will cause a run-time error.

A possible implementation for the .hpp file might be:

```
class subwidget_parser final : public rconfig::xml::xml_config_parser {
 public:
  using config_type = subwidget_config;

  /**
   * \brief The root tag that all XML configuration for exploration should lie
   * under in the XML tree.
   */
  inline static const std::string kXMLRoot = "subwidget";

  void parse(const ticpp::Element& node) override;
  std::string xml_root(void) const override { return kXMLRoot; }

 private:
  const rconfig::base_config* config_get_impl(void) const override {
    return m_config.get();
  }

  /* clang-format off */
  std::unique_ptr<config_type> m_config{nullptr};
  /* clang-format on */
};
```

A possible implementation for the .cpp file might be:

```
void subwidget_parser::parse(const ticpp::Element& node) {
  /* If our subtree not in input file, nothing to do */
  if (nullptr == node.FirstChild(xml_root(), false)) {
    return;
  }
  ticpp::Element vnode = node_get(node, xml_root());
  m_config = std::make_unique<config_type>();

  XML_PARSE_ATTR(vnode, m_config, param1);
  XML_PARSE_ATTR_DFLT(vnode, m_config, param2, std::string());
} /* parse() */
```

When creating your class you *can* override:

- validate(): Does any validation of parsed parameters. Mainly used to make sure things like angles are always > 0 but < 360, for example, which is not applicable to all parsers.

3.1.4 Registering a New Parser

Depending on what controller/loop functions are going to need your parameters, you will need to register your parser the corresponding parameter repository. For example, if I create an `energy_consumption_parser` for use by d0 controllers, I would register my parser with the `d0_controller_repository` via something like:

```
parser_register<energy_consumption_praser,
                energy_consumption_config>(
energy_consumption_praser::kXMLRoot);
```

That's it!

For the general contribution workflow, see the docs over in LIBRA.

CHAPTER

FOUR

OTHER PROJECTS (IN DESCENDING PROBABILITY OF INTEREST)

- SILICON
- SIERRA
- COSM
- RCPPSW
- RCSW