



Industrial Calibration

Chris Lewis

Southwest Research Institute

clewis@swri.org





Motivation



- Push button intrinsic and extrinsic calibration with predictable accuracy.
- Unified Framework for a wide variety of calibration tasks
- Automate motion, data collection, and solving
- Package Includes
 - Extensive set of cost functions
 - Examples
 - Tutorial





Calibration Tasks

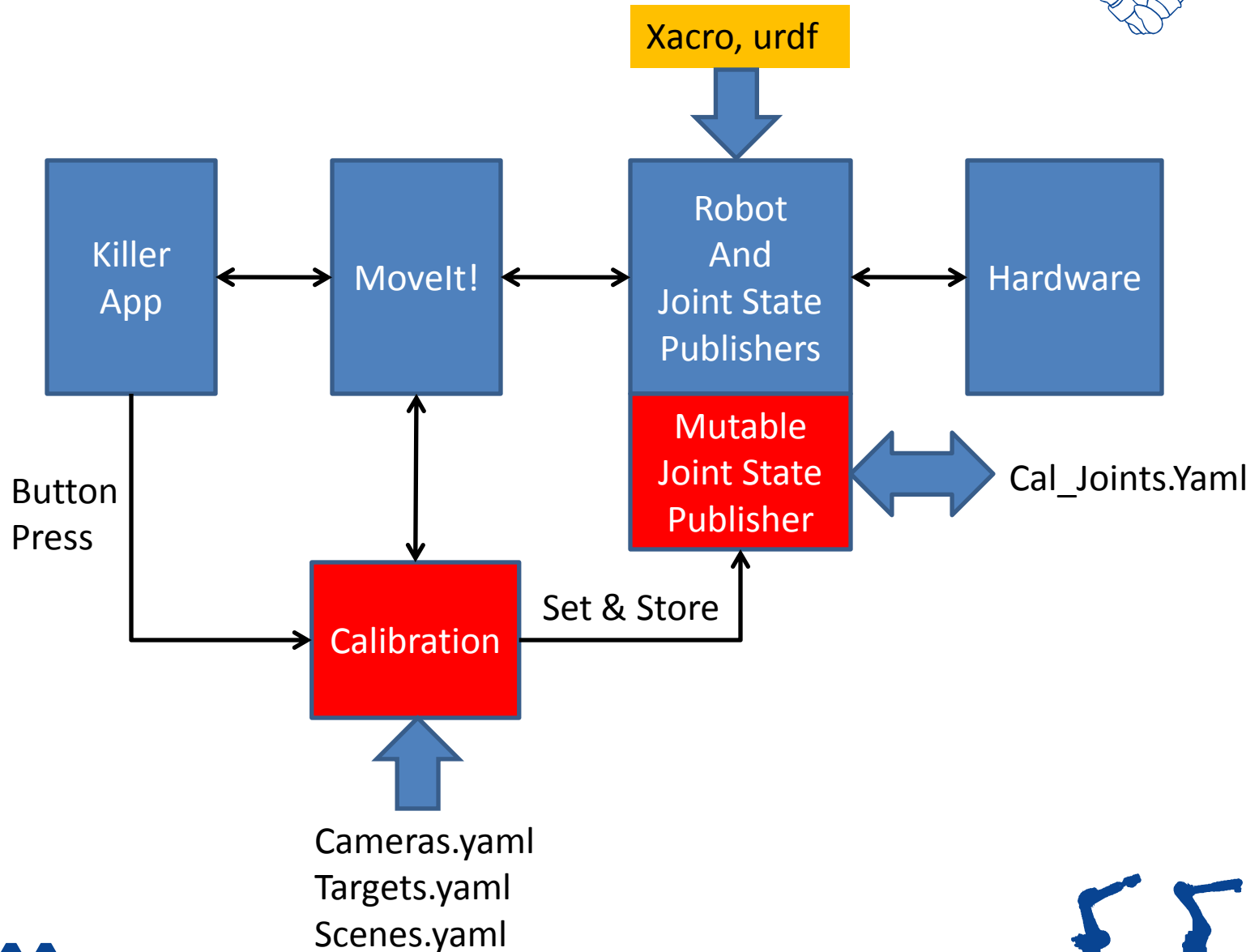


- Extrinsic calibration of an array of cameras using an array of targets
- Calibration of robots and cameras
 - Cameras observing the work cell
 - Cameras mounted on and moving with the robot
- Soon to come
 - Intrinsic Calibration
 - Stereo Calibration
- Potential Extensions: Kinematic parameters



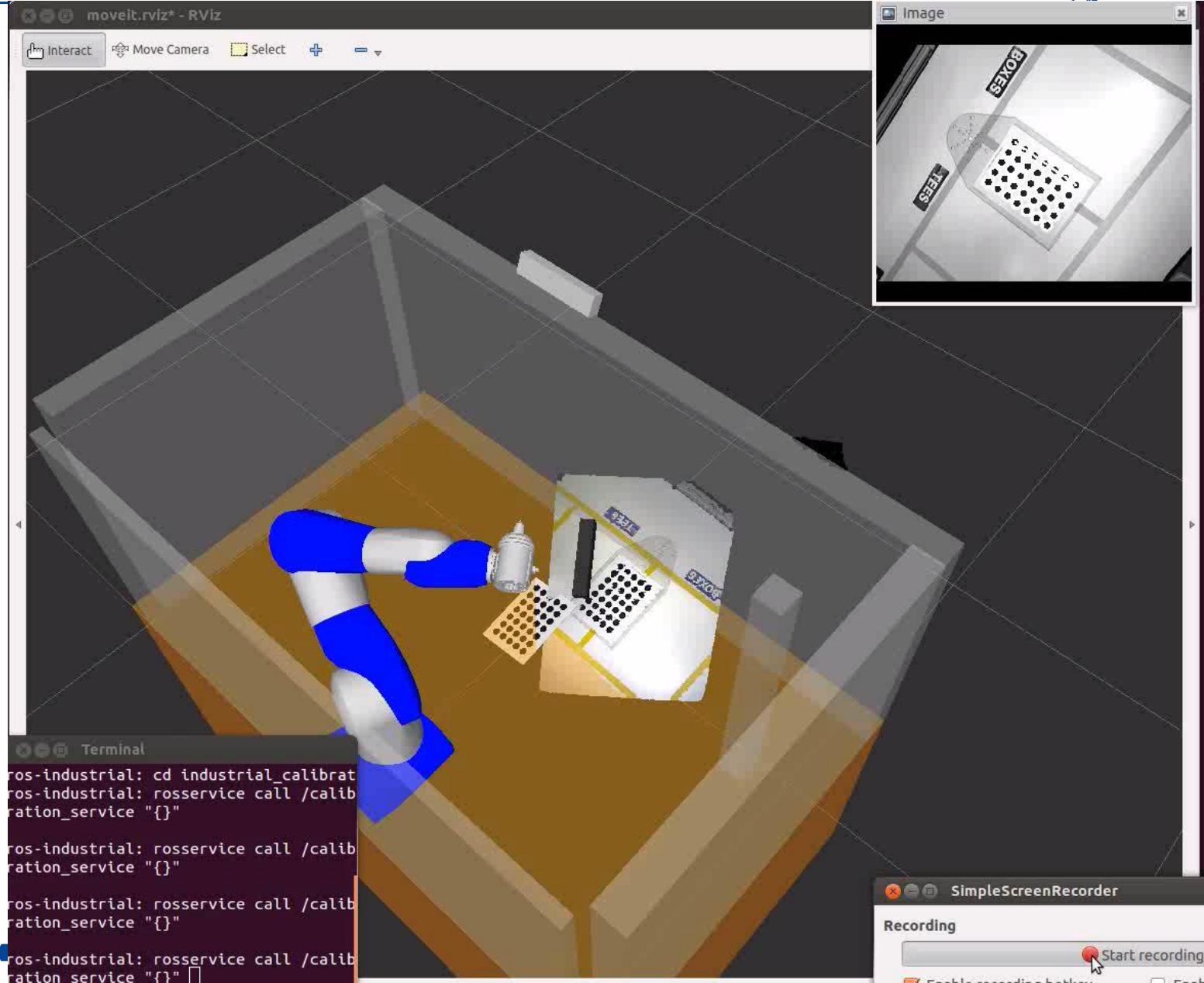


Node Architecture





Tool Mounted Camera



moveit.rviz* - RViz

Interact Move Camera Select + -

Image

BOXES

TEEB

Terminal

```
ros-industrial: cd industrial_calibrat
ros-industrial: rosservice call /calib
ration_service "{}"

ros-industrial: rosservice call /calib
ration_service "{}"

ros-industrial: rosservice call /calib
ration_service "{}"

ros-industrial: rosservice call /calib
ration_service "{}"
```

SimpleScreenRecorder

Recording

Start recording

Enable recording hotkey End





Key Aspects



- Interfaces with MoveIt! to plan and execute collision free paths between scenes.
- Collect Sufficient Statistics and Geometry
- Interfaces with TF to capture kinematics
- Results immediate and persists with re-launch
- Computes:
 - 6Dof transform between tool and camera
 - 6Dof transform between robot base and target
- Results: Consistent 3D data regardless of robot pose





Calibration Primer



- Maximize the consistency of a collection of observations with the physical model of those observations by minimizing SSE (re-projection error).
- Compute the Cost of Each Observed Point
 - Project points defined in target's frame into camera coordinates
 - Project camera points into image
 - Compare to observation

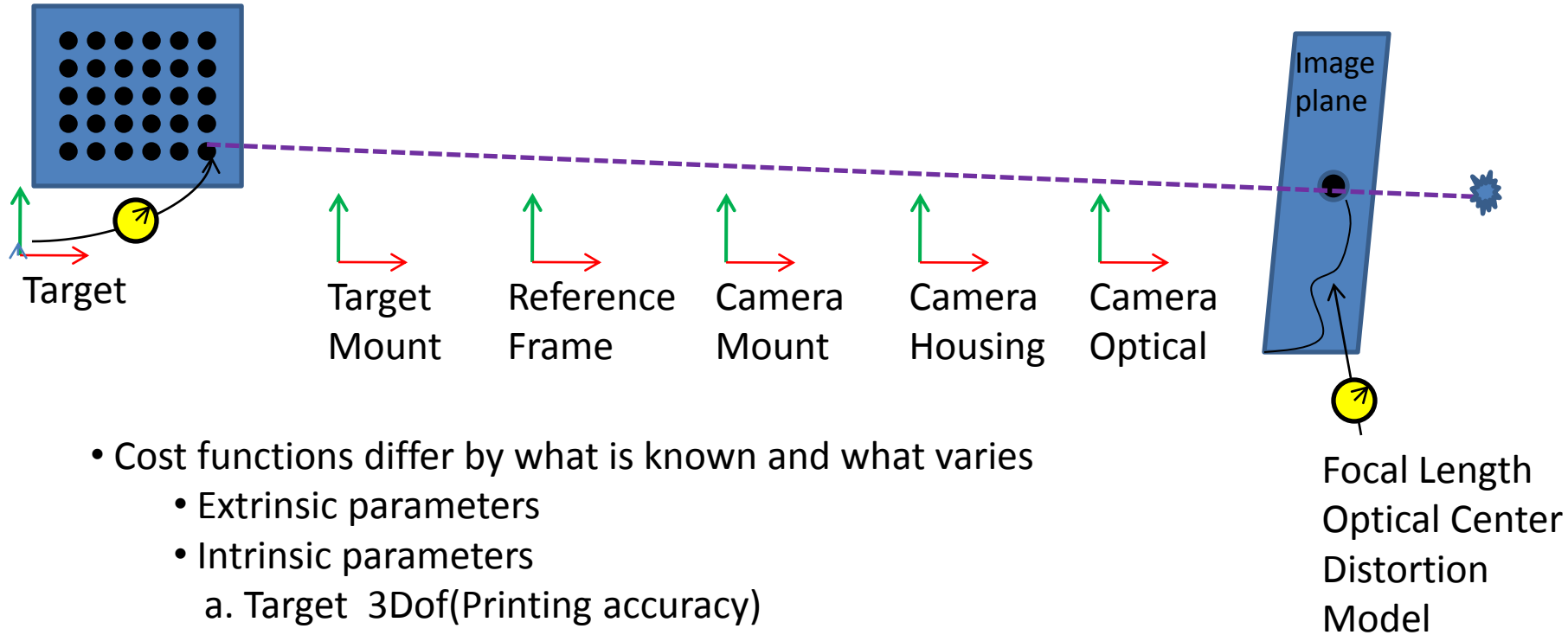




Projection



Constants and Parameters



- Cost functions differ by what is known and what varies
 - Extrinsic parameters
 - Intrinsic parameters
 - a. Target 3Dof(Printing accuracy)
 - b. Camera, typically 9Dof
- A robot might hold and move the camera or the target.
- One may want to refine the location of the device doing the holding.
- Cost functions may be written to cover any situation.





Calibration Process



- Collect Observations
 - For each static scene
 - Wait for scene trigger
 - Collect Transform Information
 - For each camera in scene
 - For each target
 - » Look for target in ROI
 - » Add observations to list
- Add each cost to a Ceres problem
- Problem.solve()
- Output Results
 - Set and store on the mutable joint state publisher
 - Write new static transform publishers





What's Special?



- Ceres
 - Best Optimization Package Ever
 - Computes analytic Jacobians from templated cost functions
- Scene Triggers
 - MoveIt!
 - Joint Value Move
 - Cartesian Move
- Camera Observers
 - Circlegrids
 - Checkerboards
 - AR Tags
- Transform Interfaces
 - TF
 - Mutable Joint State Publisher
 - Calibration Results are immediate and persist with subsequent launches
- Library of Ceres Compatible Projection Cost Functions⁹³

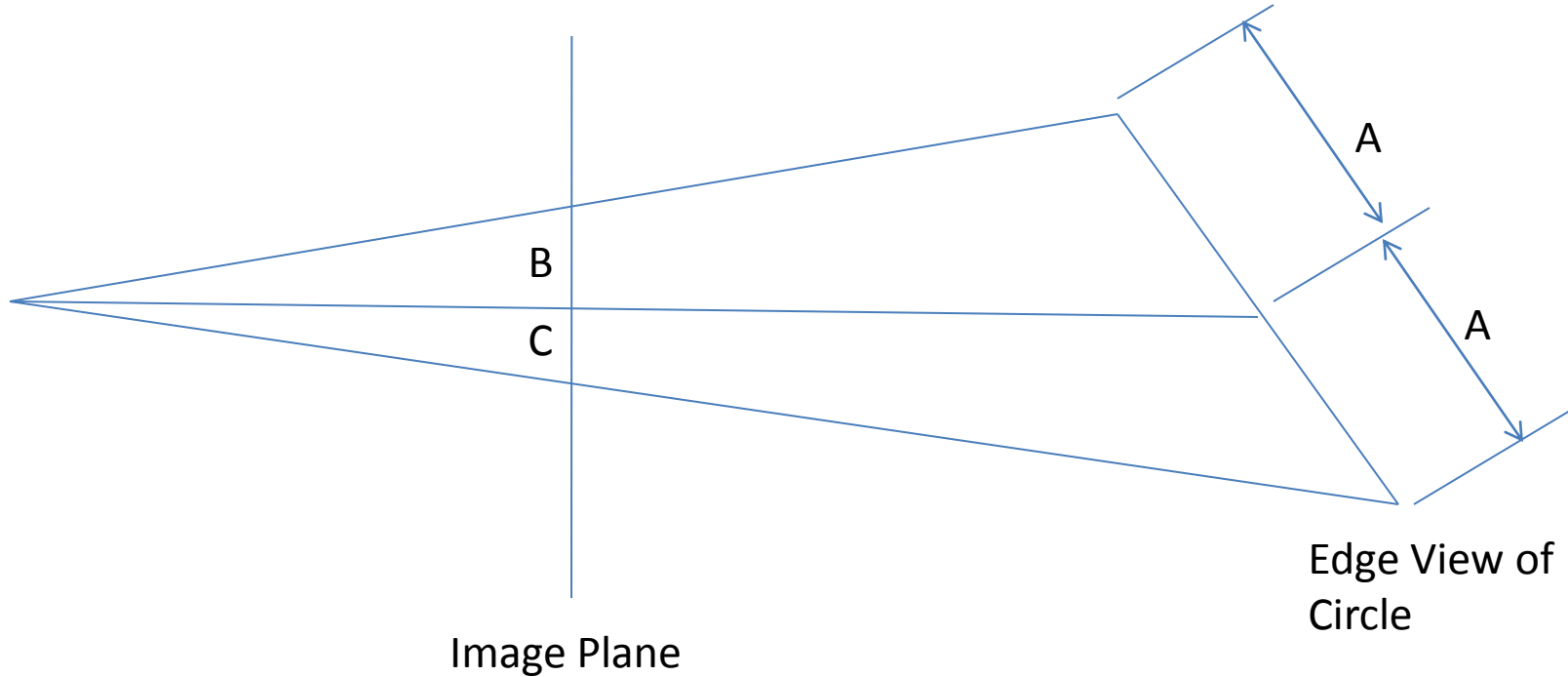




Geometry of Viewing Circles



- The center of an observed ellipse is NOT the projected center of the circular disc. $B > C$





Best Practices



- Take a Statistically Significant Set of Static Images.
 - No Motion Blur
 - Image Noise
- Cover Region of Interest
 - all the way to the edge, extrapolation is much worse than interpolation
 - Move robot to kinematic extents
- Don't Tilt Your Targets, or Use a Cost Function that Accounts for the Offset
- Don't Count on Residual Error Alone as an Indication of Accuracy
 - High residual error is bad
 - Low does not guarantee accuracy
- Use high quality targets
 - NIST Traceable if Possible
 - Calibrate target if possible
 - Lay as flat as possible





Two Asus Sensors Single Shot

The screenshot displays the ROS Industrial interface. On the left, two camera feeds are shown in 'Image' windows. The top feed shows a white perforated metal screen in a room. The bottom feed shows a similar scene from a different angle. On the right, a 3D point cloud visualization of the scene is shown. The point cloud is rendered in white and grey, with a red rectangular object on top. The interface includes a toolbar with 'Camera', 'Measure', '2D Pose Estimate', '2D Nav Goal', and 'Publish Point' buttons. At the bottom, a terminal window shows the following commands and output:

```
vecow@Vecow: ~/catkin_ws/src/10.2048
vecow@Vecow:~/catkin_ws/src/10.2048
8/calibration/launch$
vecow@Vecow:~/catkin_ws/src/10.2048
8/calibration/launch$ roscore
vecow@Vecow:~/catkin_ws/src/10.2048
8/calibration/launch$
```

Below the terminal, there are three input fields for timing information:

Elapsed: 180.86	Wall Time: 1410196601.90	Wall Elapsed: 181.48
-----------------	--------------------------	----------------------





Try Some Examples



- https://github.com/ros-industrial/industrial_calibration_tutorials.git
- `calibrate_from_images.launch`
 - Images Included
 - 8 Asus Cameras 4 on tower on either side of target
- `camera_scene_cal.launch`
 - 2 live Asus, as shown in previous video
 - Add your own cameras,
 - Swap in your own target





Where Are We?



- A few working examples
- Rough tutorials
- Lots of code not covered by gtest
- Experimental
- Needs
 - Hybrid Target Detection
 - Orientation Aware
 - Target ID Aware
 - GUI for creating yaml files

