Enhancing PX4's EKF2 Replay Module for Deterministic Integration Testing

Brian Fairservice - Software Engineer, KEF Robotics
Kerry Snyder - CTO, KEF Robotics

Presentation Overview

Integrating a visual navigation system with PX4

EKF2 Replay - What is it? Why is it helpful?

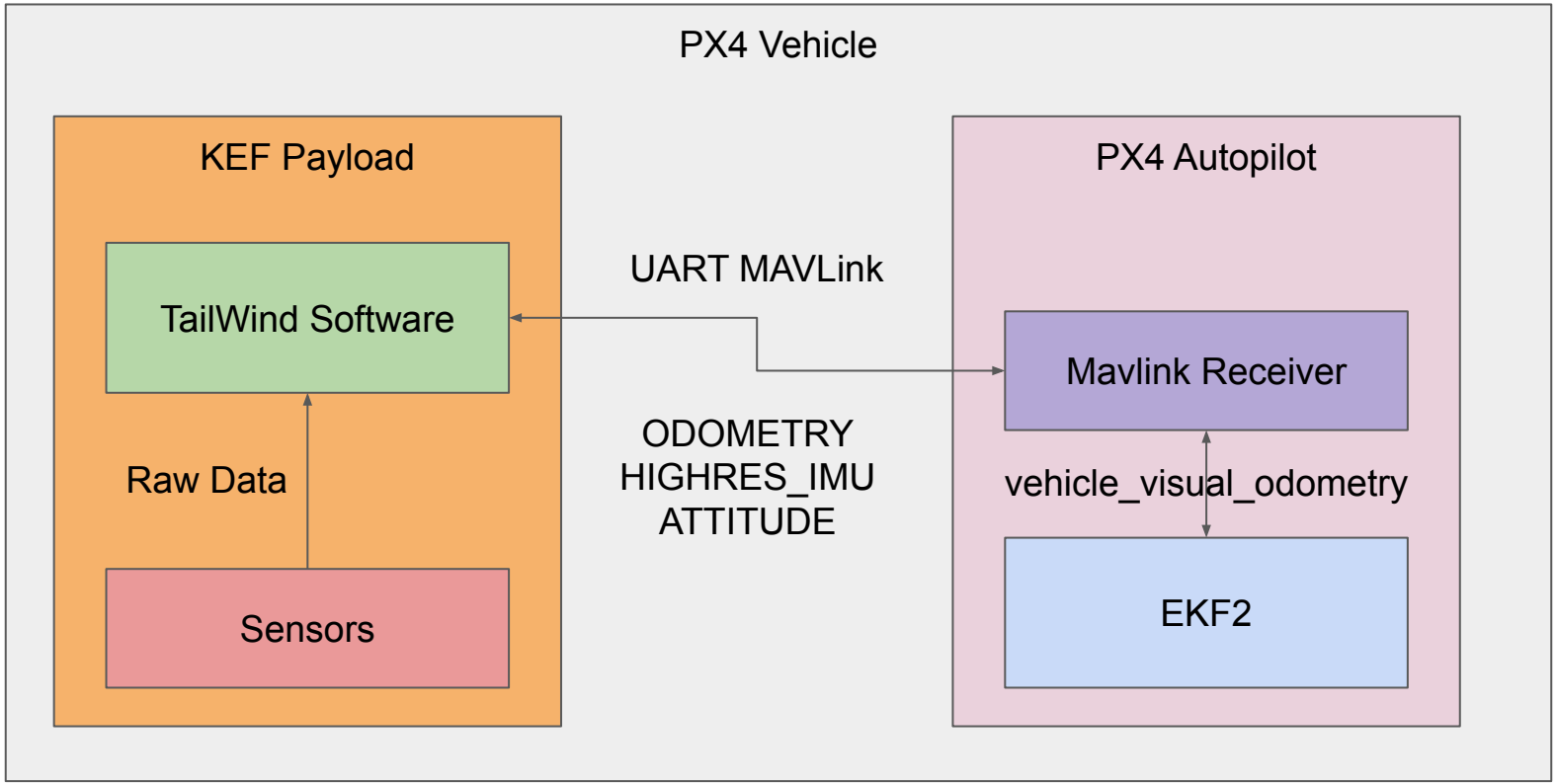
How to modify the EKF2 replay system for deterministic integration testing with a visual navigation system

Results

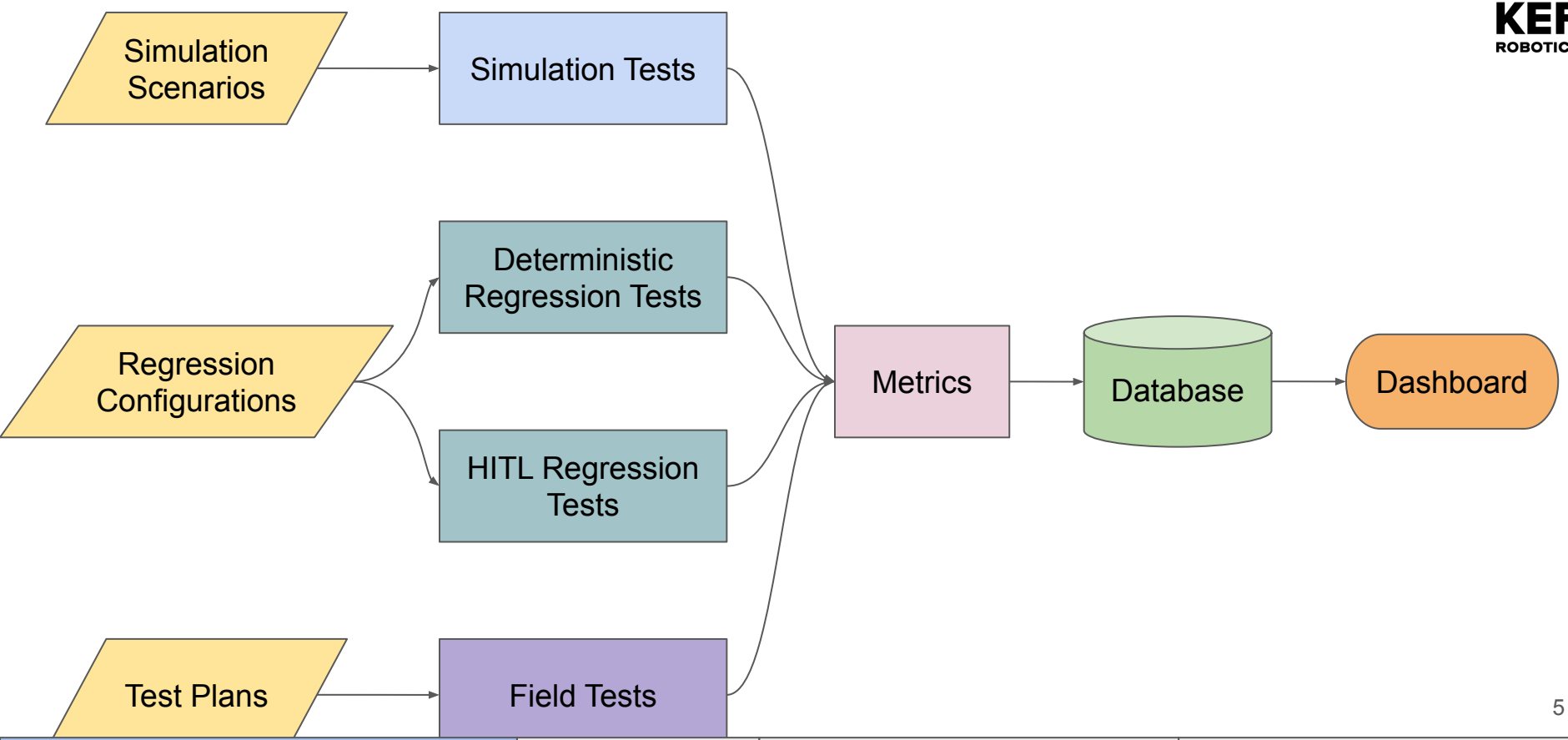
PX4 Visual Navigation and Integration Testing

Visual Navigation in PX4

"Attachable Vision Based Piloting for PX4 Vehicles - PX4 Dev Summit, 2023



Visual Navigation Software Testing

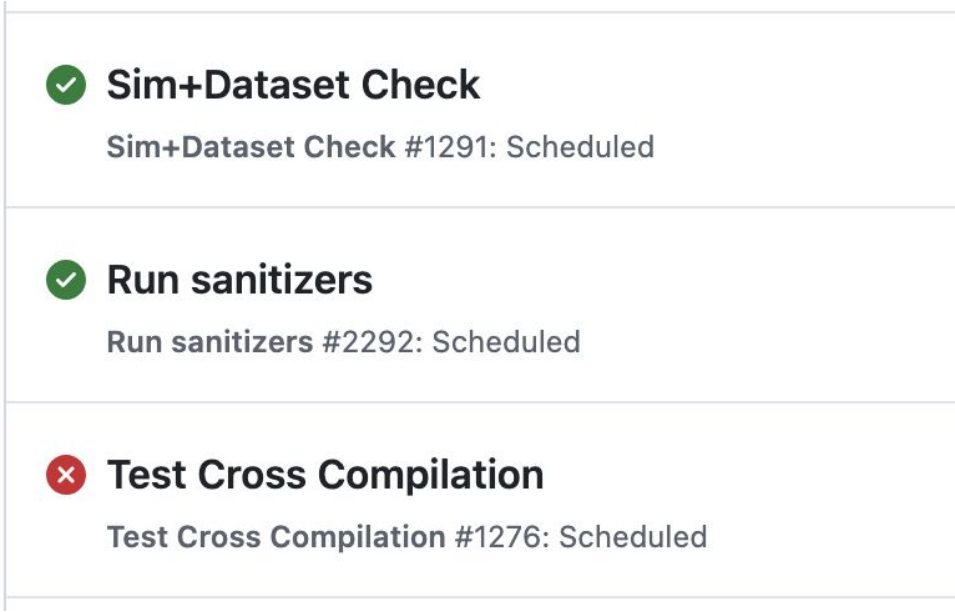


Why Determinism?

Flaky or non-reproducible Continuous Integration is a non-starter

Determinism also simplifies performance benchmarking for algorithmic changes

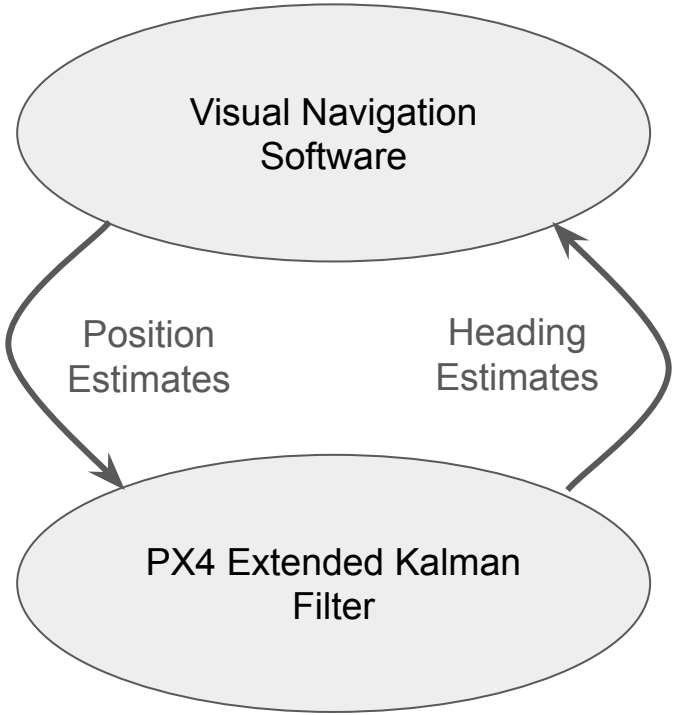
Not realistic to compute limited real time operation



Why Integration testing

In certain configurations, we have a mutual dependence between our navigation software and the PX4 EKF2:

- Magnetometer Fusion
- Fixed wing dead reckoning
- Switching between GPS and GPS-denied modes



EKF2 Replay

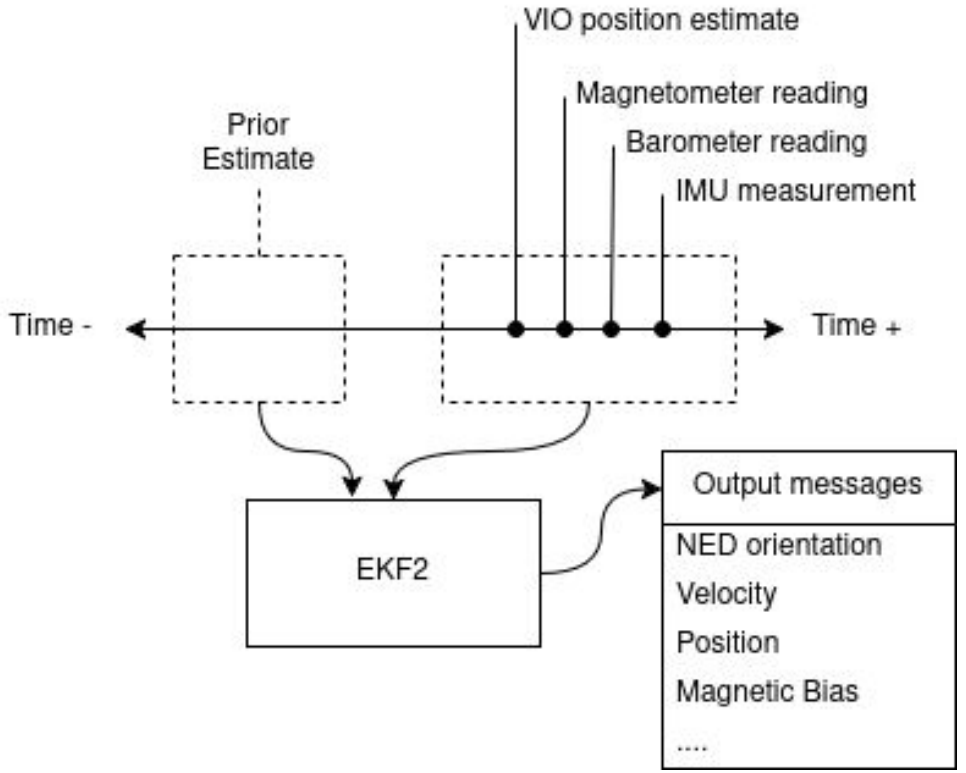
What is the EKF2 module?

The extended Kalman filter (EKF) is an algorithm for estimating the state of some system given a set of time stamped measurements and the EKF's prior estimates.

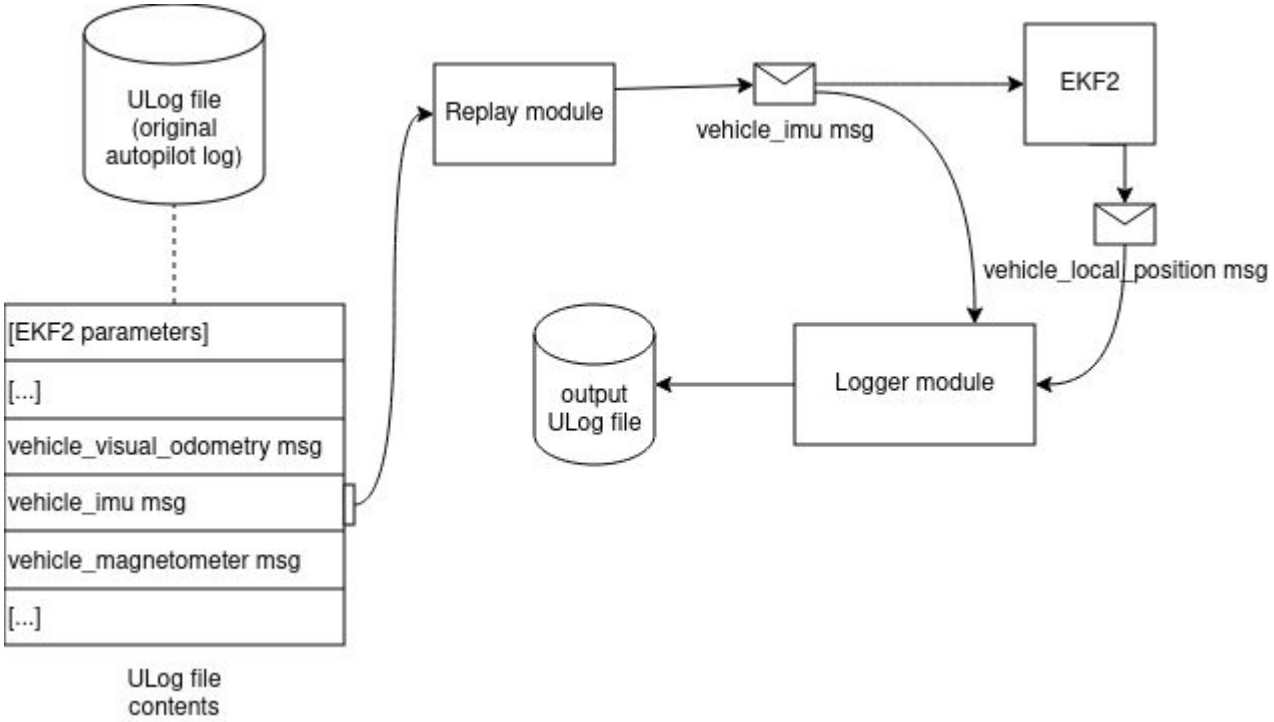
PX4's EKF2 module is an extended Kalman filter used to estimate the state of an aircraft.

The EKF2 modules subscribes to and publishes uORB messages.

It's behavior is controlled by a number of parameters, such as EKF2_GPS_CTRL, which controls how the GPS measurement should be used.



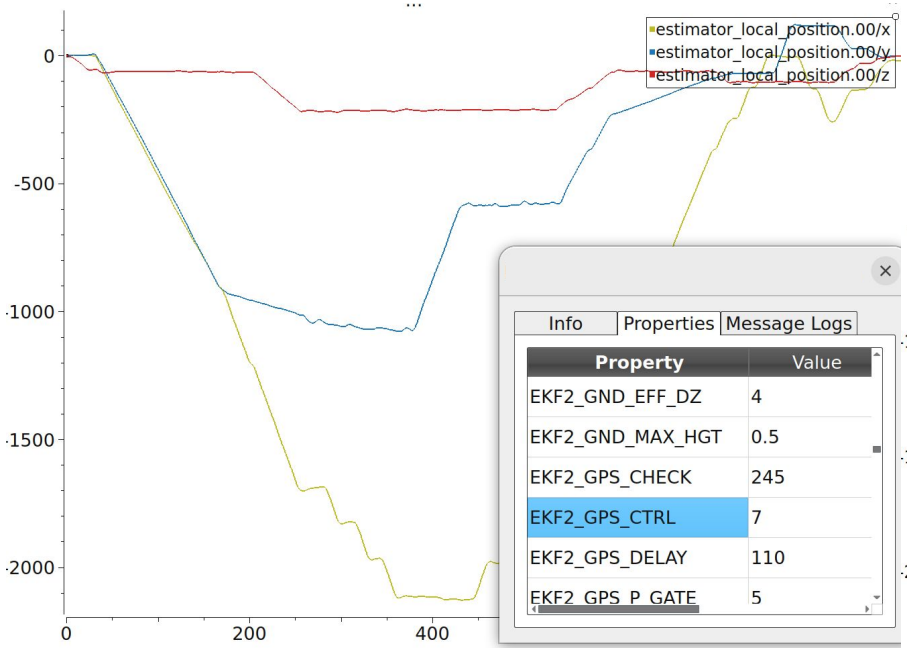
What is the EKF2 replay module?



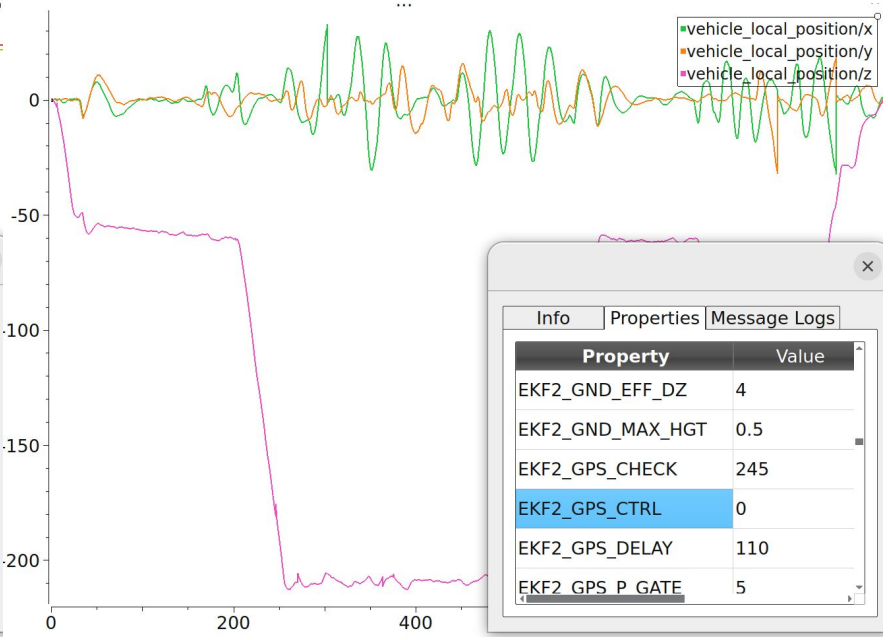
How to run PX4's EKF2 replay

```
#!/bin/bash

# The documentation says this needs to be an absolute path
export replay=$(realpath ./19_52_48.ulg)
export replay_mode=ekf2

# Override px4 parameters
echo "EKF2_GPS_CTRL 0" > ./build/px4_sitl_default_replay/rootfs/replay_params.txt

# The output log is recorded at: ./build/px4_sitl_default_replay/rootfs/log/
make px4_sitl none
```

Example EKF2 replay results



Position in the original autopilot log with EKF2_GPS_CTRL=7



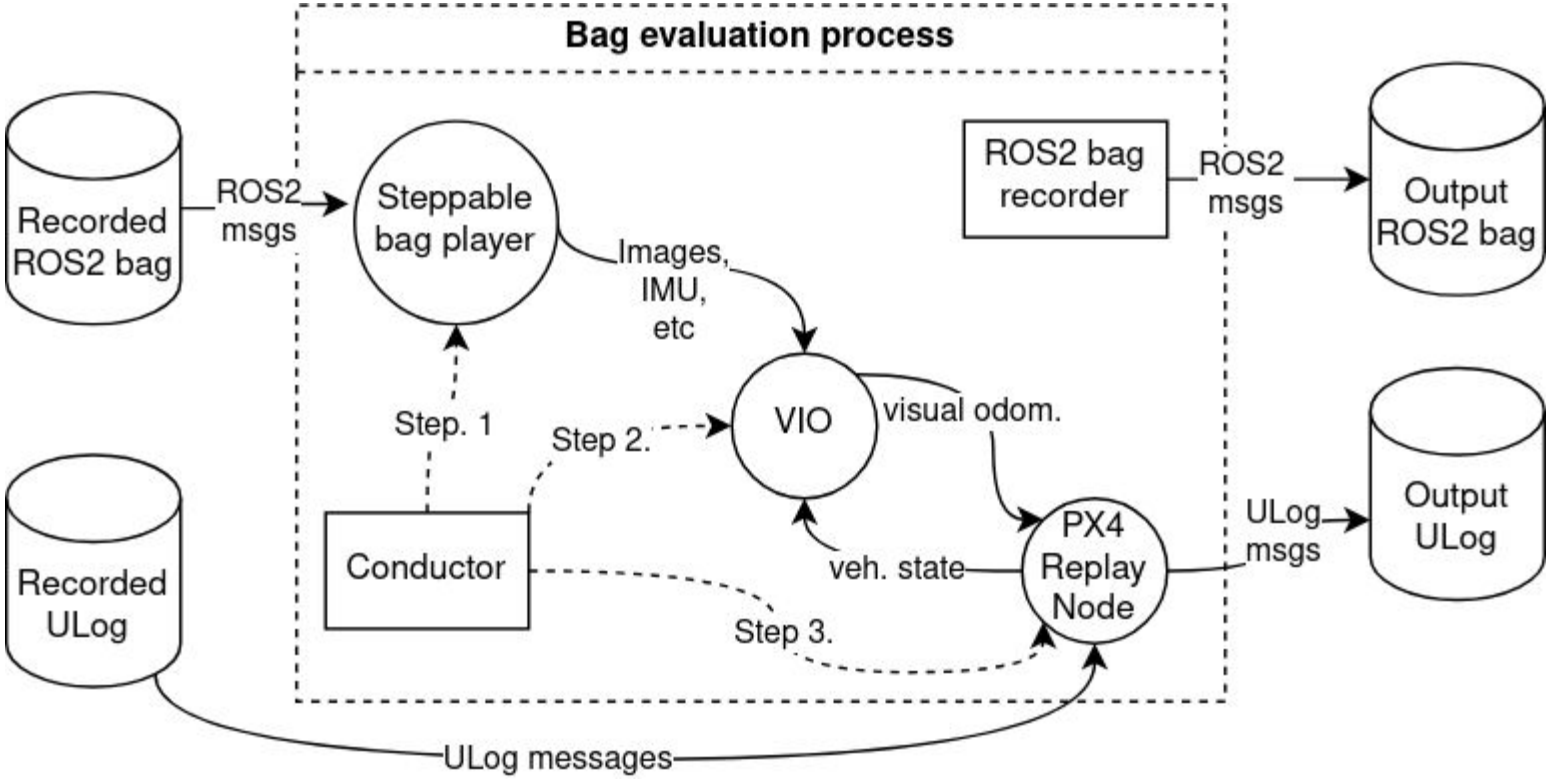
Position in the log output from the replay system with EKF2_GPS_CTRL=0

Hacking EKF2 Replay to conduct deterministic integration testing with an external visual navigation system

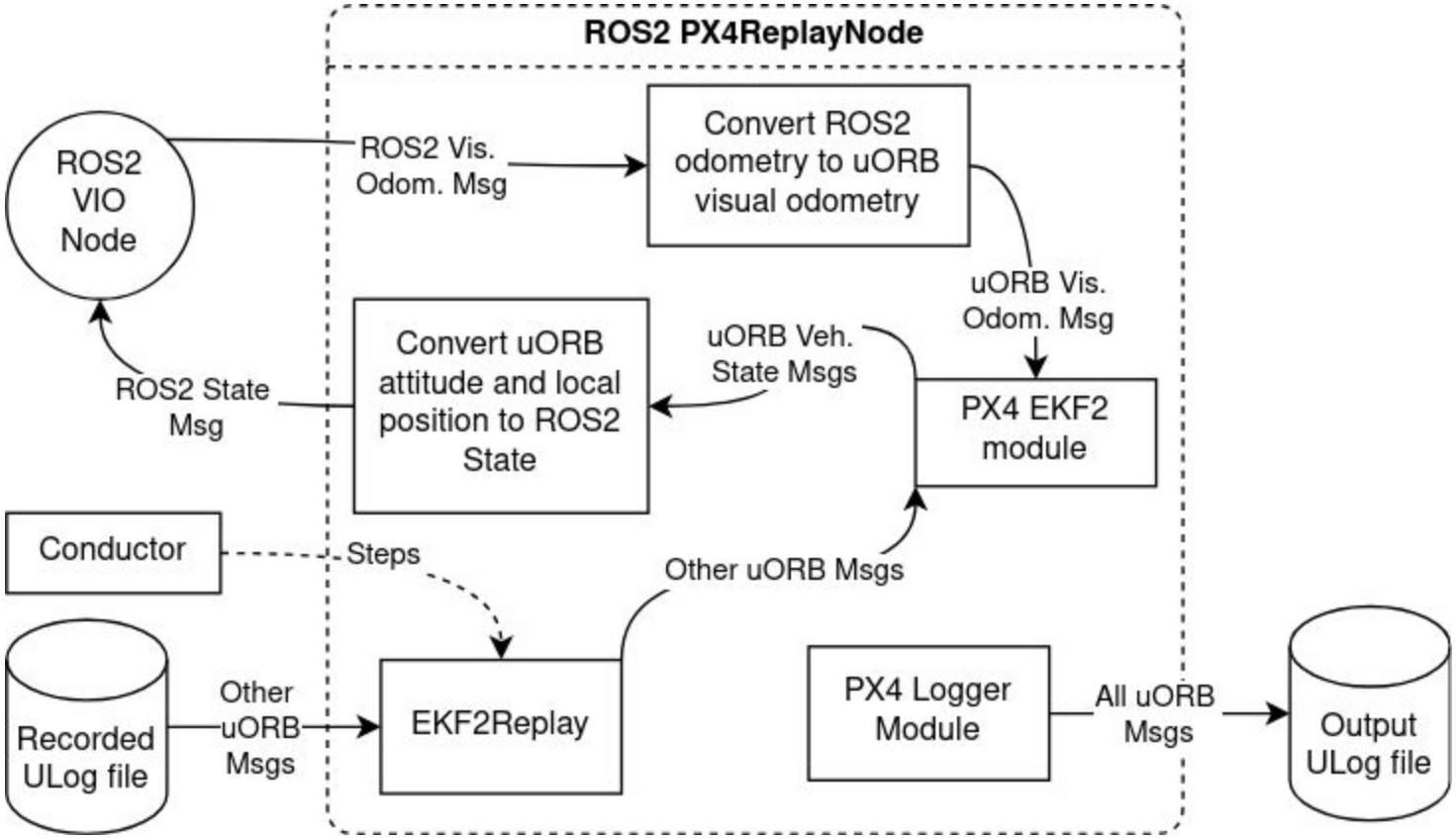
Deterministic ROS2 execution

- In order for our integration testing to be deterministic, we need to ensure that each of our ROS2 nodes is running in a well-defined order.
- We can achieve this using the following two components:
- SteppableNode
  - ROS2 node implemented in terms of a ‘runOnce()’ function.
    - A node which replays a ROS2 bag might play back the next message in the bag when its runOnce() function is called.
    - A VIO node might check if it has everything it needs to produce an odometry estimate, and if so, publish one such estimate.
- Conductor
  - Orchestrates how all the nodes run by ticking each of their runOnce() functions in turn.

Deterministic Replay System Overview



Zooming in: PX4ReplayNode



Overview of PX4ReplayNode technical challenges

To build the PX4ReplayNode, we have to link to the required PX4 modules.

PX4ReplayNode has to start and manage the logging and EKF2 PX4 modules.

We need to convert ROS2 messages (including timestamps, which is the tricky part) into their uORB counterparts, and vice versa.

The replay module has to be modified so that it can be ‘stepped’ / ‘ticked’ in sync with the rest of the system.

Linking to PX4

PX4 is not designed to be a library / submodule used by another project, it's a standalone autopilot system.

That said, PX4 builds each module as a static library that is easy to link to.

We use CMake's ExternalProject module to acquire our fork of PX4, then we link to the static libraries which get built under the CMAKE_CURRENT_BINARY_DIR.

Then we setup include paths to point into the build tree and link to the static libraries.

We also need to add some preprocessor definitions

```
ExternalProject_Add(steppable_px4
  GIT_REPOSITORY https://github.com/kefrobotics/voxl-px4
  GIT_TAG steppable-ros-replay-v1.14.0-2.0.73-dev2
  GIT_SUBMODULES_RECURSE 1

  SOURCE_DIR "${PX4_LOCATION}"
  BUILD_BYPRODUCTS "${PX4_LIB_FILES}"

  # If we don't do this, some kind of default configure step screws everything up
  CONFIGURE_COMMAND echo 'empty configure step'

  BUILD_COMMAND "${MAKE}" "CMAKE_ARGS='-DCMAKE_TOOLCHAIN_FILE=${CMAKE_TOOLCHAIN_FILE}'"

  BUILD_IN_SOURCE 1

  INSTALL_COMMAND rm -rf ${CMAKE_INSTALL_PREFIX}/include/px4
  COMMAND ln -s ${PX4_LOCATION} ${CMAKE_INSTALL_PREFIX}/include/px4
)

set(PX4_COMPILE_DEFS
  -DCONFIG_ARCH_BOARD_PX4_SITL
  -DMODULE_NAME="replay_node"
  -DORB_USE_PUBLISHER_RULES
  -DPX4_BOARD_LABEL="default"
  -DPX4_BOARD_NAME="PX4_SITL"
  -DPX4_MAIN=tests_app_main
  -D__PX4_LINUX
  -D__PX4_POSIX
)
```

Running PX4 modules in a ROS2 node

When built for Linux, PX4 modules run in their own threads.

Because PX4 modules are designed to run together on a flight computer, each module's thread is created with a small, explicitly specified stack size.

Since our PX4 modules are running in the same process as a bunch of ROS nodes, we wind up using a lot of **thread local storage** for global variables.

Glibc allocates thread local storage from the thread's stack memory, and PX4 doesn't ask for enough to accommodate all the other software it's not designed to run with.

Solution: crank up the stack space.

After that, starting these modules is as simple as calling their entry point functions.

```
static int
allocate_stack (const struct pthread_attr *attr, struct pthread **pdp,
                ALLOCATE_STACK_PARMS)
{
      // ...

      /* If the user also specified the size of the stack make sure it
         is large enough.  */
      if (attr->stacksize != 0
          && attr->stacksize < (__static_tls_size + MINIMAL_REST_STACK))
        return EINVAL;
```

```
@@ -274,7 +274,7 @@ WorkQueueManagerRun(int, char **)
274                 // It is a requirement of the  pthread_attr_setst
275                 const unsigned int page_size = sysconf(_SC_PAGESI
276                 const size_t stacksize_adj = math::max((int)PTHRE
        PX4_STACK_ADJUSTED(wq->stacksize));
    -           const size_t stacksize = (stacksize_adj + page_si
      (stacksize_adj % page_size));
277 +           const size_t stacksize = 128 * ((stacksize_adj +
```

Conversion from ROS2 messages to uORB messages

For the most part we only need to hydrate the uORB messages with data from the corresponding fields in their ROS2 counterparts.

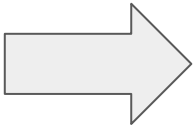
However, timestamps between the ROS2 bag and the autopilot log will not match because they were recorded on two different computers.

By matching unique positions in MAVlink messages recorded in the ROS2 bag with positions in the corresponding vehicle_visual_odometry message received on the PX4 side, we can calculate an offset from ROS2 time to PX4 time.

We perform this timestamp matching when the PX4ReplayNode initializes.

Modifying the replay module for deterministic stepping

```
void Replay::run() {
  replay_file = open_autopilot_log();

  while(!should_exit() && replay_file) {
    next_message =
peek_next_message(replay_file);
    publish(next_message);
  }

  clean_up();
}
```

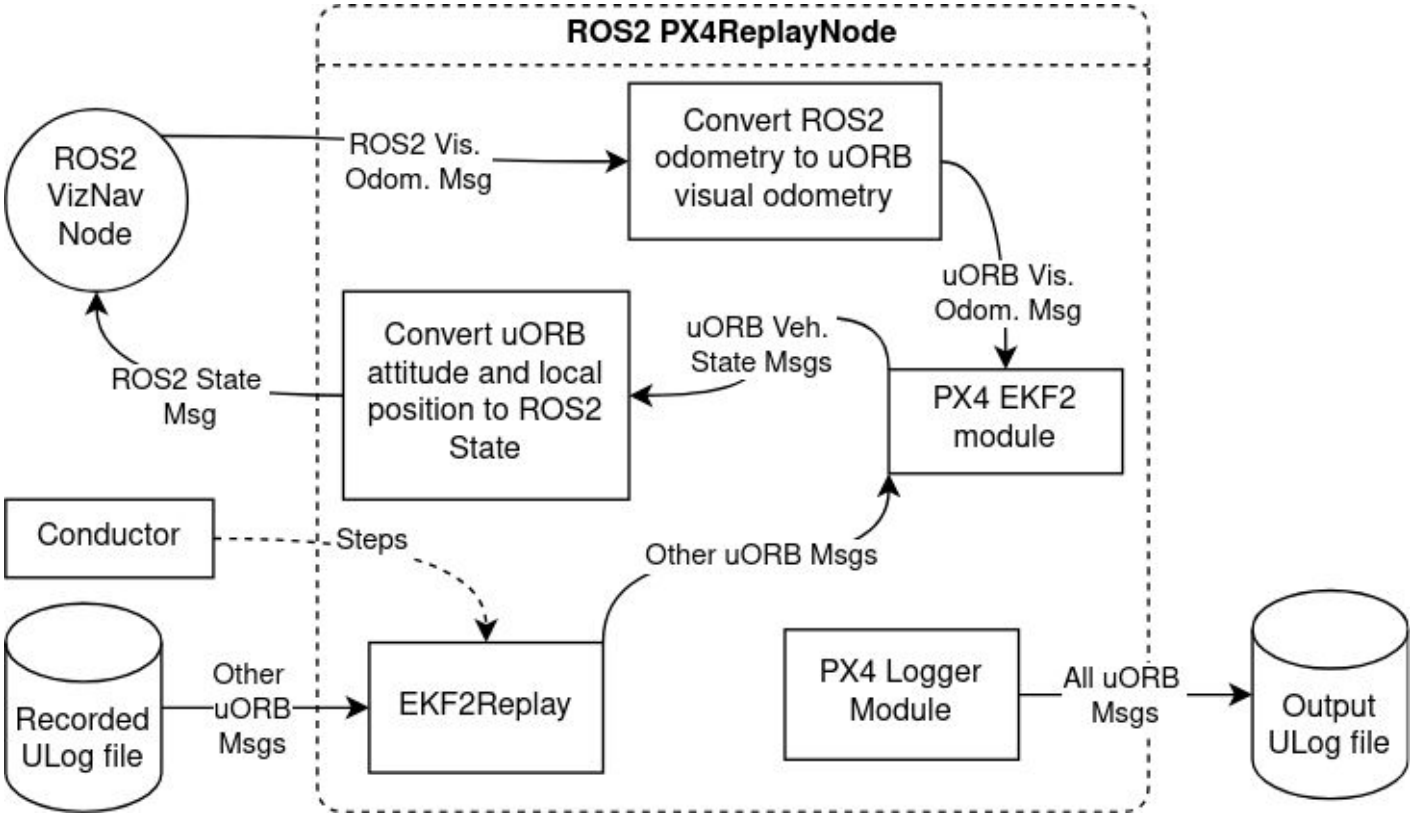


```
void Replay::setup(ignored_subs) {
  this->replay_file = open_autopilot_log();
  this->ignored_subs = ignored_subs;
}

void Replay::runTo(up_to_time) {
  while(!should_exit() && replay_file) {
    next_message =
peek_next_message(replay_file);

    if(next_message.timestamp >= up_to_time) {
      break;
    } elif(!this->shouldIgnore(next_message)) {
      publish(next_message);
    }
  }
}

void Replay::finish() {
  clean_up();
}
```

Pseudo code of the existing Replay::run() function

Pseudo code for the ‘steppable’ Replay::runTo()

Review: How the PX4ReplayNode works



Example of launching visual navigation with an accompanying PX4ReplayNode

```yaml
evaluation_bags:
- bag: ./path/to/the-ros2-bag
  truth_topic: /state/px4_gps
  kef_launch:
    nodes:
      visual_navigation_node:
        overrides:
          down_calibration: ir-cal.yaml
          mode: thermal
      px4_replay_node:
        overrides:
          ulog: ./path/to/the-autopilot-log.ulg
          autopilot_overrides:
            EKF2_GPS_CTRL: 0
            EKF2_HGT_REF: 0
            EKF2_EV_CTRL: 1
```

Results and Next Steps

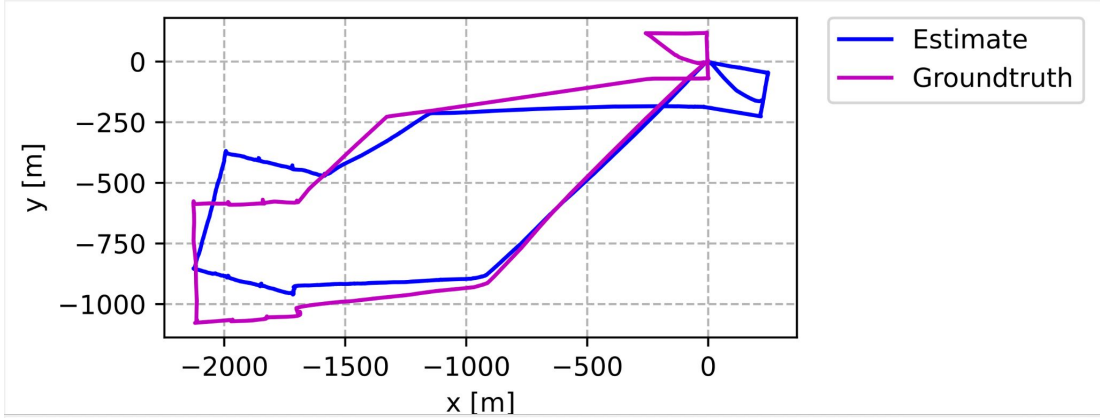
Case Study: EKF2_EVP_NOISE and Heading Fusion

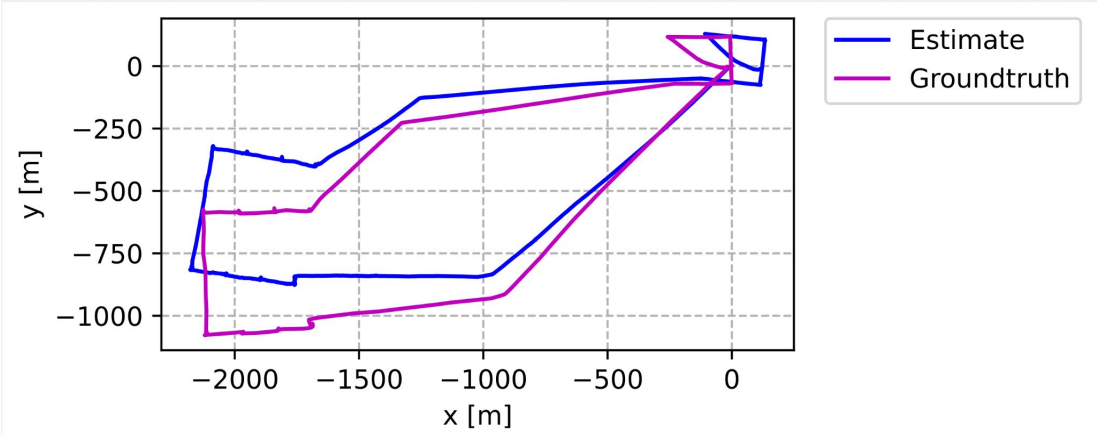
Leveraging PX4 Magnetometer fusion to improve visual-inertial odometry heading consistency

Sensitive to EKF2 noise configuration for relative input weights

Top: EKF2_EVP_NOISE: 0.3m

Bottom: EKF2_EPV_NOISE: 3.0m





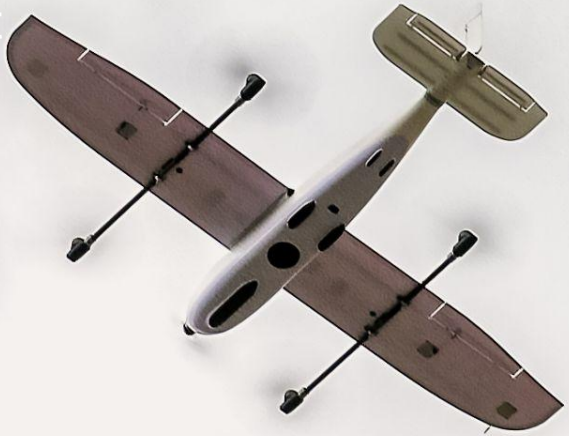
Next Steps

Leverage uXRCE-DDS to enable integration without direct linking and get closer to "test how you fly"

Integrate PX4 HITL into visual navigation HITL tests

Improve time handling to test timeout and failsafe behavior

Upstream?



Patch Link

https://www.kefrobotics.com/careers

Questions?