Learning from demonstration for head motion

Authors: ammarh@
Self Link: go/proxy-head-lfd
Visibility: Confidential.
Originally Proposed: 2021-04-02

Objective

Proxy robots currently navigate around by perceiving their environment using point cloud data obtained through a fixed mounted CBr lidar sensor as well as a stereo camera pair mounted on a pan-tilt head. AutoLook (go/proxy-autolook) enables an intelligent framework where the head is more actively involved in observing areas of most interest to the planner. Through AutoLook the robot is able to complement the observations from the CBr to fill in the gaps caused due to blindspots, data sparsity or general observation staleness. This has been successfully deployed and is actively running on all Proxy robots during navigation since 2019.

This proposal extends the head control policies within AutoLook per the Phase 3 rollout outlined in go/proxy-autolook. This phase attempts to unify multiple control policies that are currently triggered by just the base planner into a unified control policy.

Requirements

Proxy AutoLook was initially meant to just serve the base planner. However, given its utility while navigating it makes sense to serve other use cases, for example help the people detection & tracking system. Here are some requirements for this project:

- A unified control policy that takes input the state of the base planner as well as the object tracker and possibly other modules.
- The AutoLook policy determines how the head should move and what observations it must service.
- The policy must be data driven rather than with hand coded heuristics.
- The head should move in a human friendly & intuitive manner pursuant to HRI. In other words, it should constantly stare at someone or weirdly jitter its head around.

Out of scope

- This proposal does not design and build a remote teleoperation system. It assumes the existence of one that may be leveraged for collecting human demonstrations.
- Problems related to true occlusions, such as stuff hidden behind a desk, that is unobservable in any of the robot sensing modalities are also not solved in this design. It instead, tackles blindspots in one that can be complemented through observation in another.

Background

Most of the background on AutoLook such as motivation, design and implementation are outlined in detail in go/proxy-autolook. The current implementation code can be found here. It is based on switching between two different control policies that are triggered by the robot base planner in the auto_look_inputs proto.

Design ideas

This doc proposed a learning from demonstration based control policy to guide the motion of the head into a unified cost framework that can take inputs from multiple sources such as base planner, object tracker etc.

Learning from Demonstration (LfD) (also known as Imitation Learning) is a paradigm for enabling robots to autonomously perform new tasks. Rather than requiring a slew of heuristics or manual programming that decomposes a task to get the desired behavior, LfD takes the view that an appropriate robot controller can be derived from observations of a human's own performance. It tries to imitate how a human would have performed the given task given the inputs & observations from the environment, by leveraging a large repository of past human demonstrations in similar situations. Traditionally a human programmer would have to reason in advance and code the head control policies that are capable of responding to any situation the robot may face, no matter how unlikely. This is an active and ongoing area of research with very promising results.

By leveraging human demonstrations the robot simply relies on a human teacher to learn from. This makes the entire AutoLook system data driven rather than hand tuned cost heuristics. A happy side effect of this approach is that the head motion becomes a lot more human and HRI friendly, since the robot after all learned from a human.

The major challenge of this approach is having the ability to collect large amounts of meaningful & representative human demonstration of how the robot head should move for optimizing seamless navigation. This could be done in multiple ways:

(a) Human operator walking with the robot as it is navigating and guiding the head using a joystick controller.
(b) Remote human operator sitting in some base station viewing the head observations + other sensor data on a monitor and navigating the robot based on that.
(c) Remote human operator sitting in an immersive experience such as a VR headset where the environment is created through the online sensor and head observations.

Approach (a) is fairly limiting and will not yield faithful human demonstration given how hard it is to actively control the head while walking around. Proxy is well suited for this LfD based AutoLook head controller since there is a major investment in building remote teleop capabilities, which at minimum would be approach (b) while targeting (c). A human wearing a VR HUD headset provides amazing and easy to collect demonstrations to learn an intuitive head control policy from. These trajectories as labels and the planner, tracker states as inputs can then be fed into a simple supervised learning to regress on the best possible direction where the head should look at.

Alternatives considered

This design proposal strongly leans away from manually created and tuned heuristics. These knobs, while easy to create at first, by decomposing a few different scenarios eventually become intractable given the diversity of scenarios that the robot encounters.This is evidently clear with the hand tuned geometric heuristics that the current navigation system relies on.

Another alternative that this design proposal choses to discard is having a finite state machine that arbitrates what control policy dictates the head motion. This is the design of the current AutoLook system where the `BaseLocalPlannerToAutoLookProto` is used to switch between LOOK_AHEAD vs the CLEAR_UNKNOWN policy. However such a system is not easy extensible to take in inputs from other sources and becomes limiting for future use cases. It is also not driven by data but instead needs to be constantly reprogrammed.
