# Building a Cloud Control Framework for the TurtleBot

Leif Christiansen[2], Supun Kamburugamuva[1], Geoffrey Fox[1]
[1] Indiana University
[2] Lewis & Clark College

## Introduction

The availability of Internet connections and low manufacturing costs have led to a surfeit in *smart objects*, simple devices consisting of a CPU, memory storage, and a wireless connection.[3] When equipped with sensors and actuators such objects may provide exciting advances in areas including exploration, surveillance, and emergency response; if a suitable framework can be deployed to process sensor data and return control messages.

For such sensors, cloud processing has been shown to be advantageous over embedded processors due to numerous factors: the ability to easily draw from vast stores of information, efficient allocation of computing resources, a proclivity for parallelization, the ability to handle large amounts of information, operation of multiple units simultaneously, etc. [1]

One sensor that has experienced great popularity, both in academia and among the public, is the Microsoft Kinect. The Kinect (shown in Fig. 1) is a fairly cheap and extremely useful sensor with a host of applications ranging from physics to physical therapy. As a result, there is a wealth of information and resources surrounding the Kinect.

Our project expands upon previous work on the Kinect, creating a control framework for the TurtleBot (Fig. 2) that performs all data processing on the cloud.


Fig. 1: Microsoft Kinect, without casing

A Mobile Base and Power Board
B 3D Sensor
C Computing
D TurtleBot Hardware

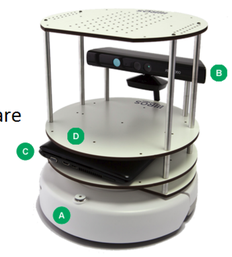


Fig. 2: Willow Garage's TurtleBot

## Components

**libfreenect:** An open-source Kinect driver produced by OpenKinect.

**IoTCloud2**: A sensor-centric middleware developed to bring sensor data to the cloud. IoTCloud2 consists of a controller for holding sensor specific state information and a message broker (RabbitMQ was used in our project) for passing state updates, data messages and control messages.

**RabbitMQ**: A message broker based on the Advanced Message Queuing Protocol (AMQP) model. TCP is used for delivery. [5]

**Apache Storm**: A distributed real-time computation system. Apache Storm consists of spouts and bolts, nodes used to stream and process data respectively. The overall network of these nodes is referred to as a topology. [6]

## Reading Distance from the Kinect

Each frame is sent as a bytebuffer containing 307,200 10-bit disparity values, 1024 indicating an unreadable point. The disparities are converted into meters using an algorithm given by Stéphane Magnenat. [4]

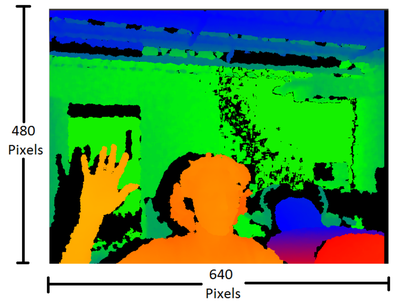


Fig. 3: Sample frame from the depth camera

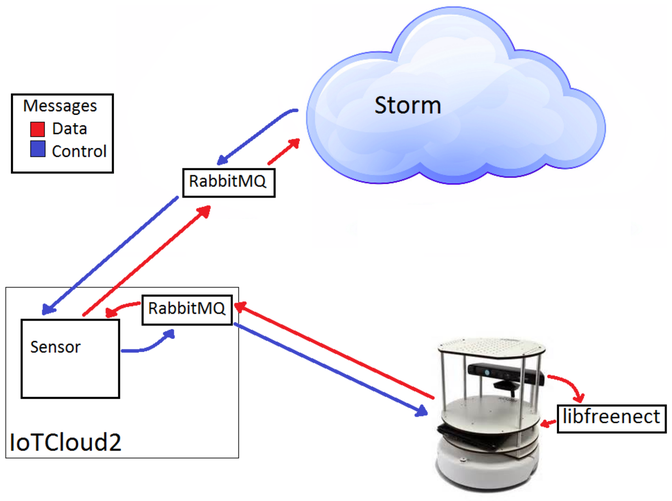


Fig. 4: The architecture of our project

## Compression

In order to process data in real-time the transmitted frames needed to be compressed.

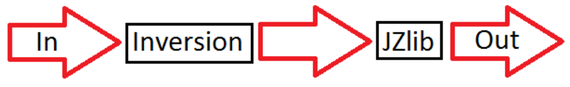


Fig. 5: Our two-stage compression scheme, compresses 10:1 in 10ms

**Inversion**: Mehrotra et al's inversion algorithm encodes multiple values the same, taking advantage of the Kinect error.
**JZlib:** A Java implementation of Zlib, a compression library that intelligently switches between LZ77 and Huffman coding.

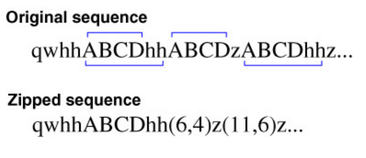


Fig. 6: Example of LZ77 compression

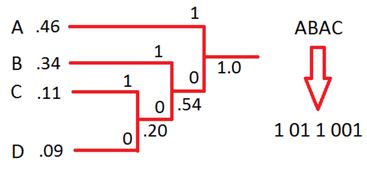


Fig. 7: Example of Huffman coding

## Processing/Functionality

An Apache Storm topology was deployed on the cloud implementing the follower program.

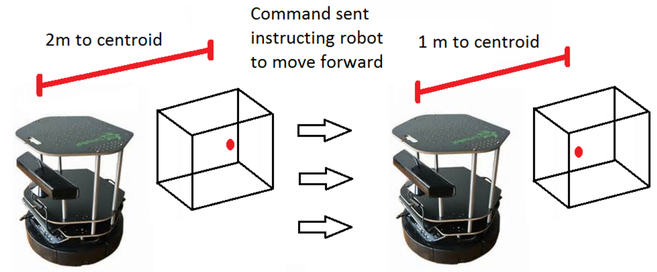


Fig. 8: The follower calculates an average point (centroid) from the TurtleBot's FoV and seeks to maintain a constant distance

## Conclusion

Our framework successfully processes Kinect frames on the cloud and returns control messages to the TurtleBot in real-time. This is achieved through our two-stage compression scheme coupled with effective message passing technologies. We are confident that this framework may be scaled to process and control a large number of TurtleBots.

## Future Work

In order to test the scalability of our framework we are currently in the process of developing a program to simulate numerous sensors running simultaneously.

The framework is also being tested so as to determine the optimal stage for compression, be it on the TurtleBot computer or IoTCloud2, and identify areas most in need of improvement.

## References

[1] G.C. Fox, S. Kamburugamuve, R.D. Hartman, Architecture and measured characteristics of a cloud based internet of things API, in International Conference on Collaboration Technologies and Systems, (Colorado, USA, 2012).
[2] S. Mehrotra, Z. Zhang, Q. Cai, C. Zhang, and P. A. Chou, Low-complexity, near-lossless coding of depth maps from kinect-like depth cameras, in Proc. IEEE MMSP Workshop, (Hangzhou, China, 2011), IEEE, pp 1-6.
[3] IPSO White Paper #1 "Internet Protocol for Smart Objects (IPSO) Alliance"
[4] OpenKinect, *Imaging information*. [Acces 2014 June 20] Available from: http://openkinect.org/wiki/Imaging_Information
[5] RabbitMQ, *Messaging that just works*. [Accessed 2014 July 5] Available from: https://www.rabbitmq.com/
[6] Storm, *Distributed and fault-tolerant realtime computing*. [Accessed 2014 July 14] Available from: https://storm.incubator.apache.org

## Acknowledgments

Thanks to Dr. Geoffrey Fox and Gregor Von Laszewksi for their consulting and tutelage and also to the IU-SROC for making it all possible.

## Primary Contact

Supun Kamburugamuva, Indiana University, supun06@gmail.com