The **JOY**

Of Flying Robots

with **Clojure**

— Carin Meier

@carinmeier
Long Ago

In a time before
Long Ago
In a time before mobile phones
Long Ago

In a time before

mobile phones
personal computers
ordinary girl
ROBOT FRIEND
ROBOT FRIEND
dolls
ROBOT FRIEND

dolls

stuffed animals
Years Passed

software developer
companies
languages
kids
Hack Your Roomba
Hack Your Roomba

ROBOT FRIEND?
How do you talk to a robot?
LISP
(def cat "cat")
;;=> "cat"
(defn say-hello [name]
  (str "hello " name))

(say-hello "roomba")
;;=> "hello roomba"
(class "roomba")
;; => java.lang.String

(.toUpperCase "roomba")
;; => "ROOMBA"
Clojure

Dynamic

Functional

Java Interop

Concurrency

Immutable Data

Vars

Refs

Atoms

Agents
iRobot® Roomba®
Serial Command Interface (SCI) Specification
iRobot® Roomba®
Serial Command Interface (SCI) Specification

RooTooth - Bluetooth Wireless Roomba Connection
DEV-10980 RoHS

**Description:** A cable-less solution developed for controlling your Roomba. The RooTooth v2 communicates with any Bluetooth® enabled device using the Serial Port Profile (SPP). You can now tell the Roomba when to wake up, when to clean, and when to shut off. You can even remotely drive your Roomba from your computer or from the WiTilt!

This product requires that the SCI (serial command interface) has been installed on your Roomba. SCI has been installed on all Roombas manufactured after October 24th, 2005. Please verify that your Roomba is compatible by checking here.
RoombaComm

[NOTE: RoombaComm is now being actively maintained by Paul Bouchier, Jonathan Pitts & Matt Black (and occasionally me perhaps) on the Dallas Personal Robotics Group site. Check out RoombaComm's new home! This site will continue to mirror any RoombaComm updates by the DPRG folks. The DPRG site has many great projects and tutorials for building your own robots, be they Roomba-based or not. Thank you so much guys for giving RoombaComm some much-needed attention.]

RoombaCommTest

<table>
<thead>
<tr>
<th>Commands</th>
<th>Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td></td>
</tr>
<tr>
<td>power-off</td>
<td>wakeup</td>
</tr>
<tr>
<td>beep-lo</td>
<td>beep-hi</td>
</tr>
<tr>
<td>clean</td>
<td>spot</td>
</tr>
<tr>
<td>vacuum-on</td>
<td>vacuum-off</td>
</tr>
</tbody>
</table>
Roomba Demo
Backup Video

Just in case
Happy Times :)
Robot Friend?
AR Drone
Quadcopter
Two Cameras
Sonar
Can Fly it with iPhone
AR Drone Developer Guide

Contents

A.R.Drone Developer Guide
Contents

1 SDK documentation
  1 Introduction

2 A.R.Drone 2.0 Overview
  2.1 Introduction to quadrotor UAV
  2.2 Indoor and outdoor design configurations
  2.3 Engines
  2.4 LiPo batteries
  2.5 Motion sensors
  2.6 Assisted control of basic manoeuvres
  2.7 Advanced manoeuvres using host tilt sensors
  2.8 Video streaming, tags and roundel detection
  2.9 Wifi network and connection
  2.10 Communication services between the A.R.Drone 2.0 and a client device
  2.11 Differences between A.R.Drone 2.0 and A.R.Drone 1.0

3 A.R.Drone 2.0 SDK Overview
  3.1 Layered architecture
UDP Communication

(ns clj-drone.baby-steps
  (:import (java.net DatagramPacket DatagramSocket InetAddress)))

(def drone-host (InetAddress/getByName "192.168.1.1"))
(def at-port 5556)

(def socket (DatagramSocket. ))

(defn send-command [data]
  (.send socket
    (new DatagramPacket (.getBytes data) (.length data) drone-host at-port)))

(def take-off "AT*REF=1,290718208\r")
(def land "AT*REF=2,290717696\r")

(send-command take-off)
(send-command land)
(ns clj-drone.baby-steps
  (:import (java.net DatagramPacket DatagramSocket InetAddress)))

(def drone-host (InetAddress/getByName "192.168.1.1"))
(def at-port 5556)

def socket (DatagramSocket. ))

(defn send-command [data]
  (.send socket
    (new DatagramPacket (.getBytes data) (.length data) drone-host at-port)))

(def take-off "AT*REF=1,290718208\r")
(def land  "AT*REF=2,290717696\r")

(send-command take-off)
(send-command land)
UDP Communication

(ns clj-drone.baby-steps
  (:import (java.net DatagramPacket DatagramSocket InetAddress)))

(def drone-host (InetAddress/getByName "192.168.1.1"))
(def at-port 5556)

(def socket (DatagramSocket. ))

(defn send-command [data]
  (.send socket
    (new DatagramPacket (.getBytes data) (.length data) drone-host at-port)))

(def take-off "AT*REF=1,290718208\r")
(def land "AT*REF=2,290717696\r")

(send-command take-off)
(send-command land)
(ns clj-drone.baby-steps
  (:import (java.net DatagramPacket DatagramSocket InetAddress)))

(def drone-host (InetAddress/getByName "192.168.1.1"))
(def at-port 5556)

(def socket (DatagramSocket. ))

(defn send-command [data]
  (.send socket
    (new DatagramPacket (.getBytes data) (.length data) drone-host at-port)))

(def take-off "AT*REF=1,290718208\r")
(def land "AT*REF=2,290717696\r")

(send-command take-off)
(send-command land)
(ns clj-drone.baby-steps
   (:import (java.net DatagramPacket DatagramSocket InetAddress)))

(def drone-host (InetAddress/getByName "192.168.1.1"))
(def at-port 5556)
(def socket (DatagramSocket. ))

defn send-command [data]
   (.send socket
      (new DatagramPacket (.getBytes data) (.length data) drone-host at-port)))

(def take-off "AT*REF=1,290718208\r")
(def land "AT*REF=2,290717696\r")

(send-command take-off)
(send-command land)
OMG IT WORKED!!!!
clj-drone
(def commands
{
  :take-off {:command-class "AT*REF" :command-bit-vec [9 18 20 22 24 28]}
  :land   {:command-class "AT*REF" :command-bit-vec [18 20 22 24 28]}
  :emergency {:command-class "AT*REF" :command-bit-vec [8 18 20 22 24 28]}
  :spin-right {:command-class "AT*PCMD" :command-vec [1 0 0 0 :v] :dir 1}
  :spin-left {:command-class "AT*PCMD" :command-vec [1 0 0 0 :v] :dir -1}
  :up      {:command-class "AT*PCMD" :command-vec [1 0 0 :v 0] :dir 1}
  :down    {:command-class "AT*PCMD" :command-vec [1 0 0 :v 0] :dir -1}
  :tilt-back {:command-class "AT*PCMD" :command-vec [1 0 :v 0 0] :dir 1}
  :tilt-front {:command-class "AT*PCMD" :command-vec [1 0 :v 0 0] :dir -1}
  :tilt-right {:command-class "AT*PCMD" :command-vec [1 :v 0 0 0] :dir 1}
  :tilt-left {:command-class "AT*PCMD" :command-vec [1 :v 0 0 0] :dir -1}
  :fly     {:command-class "AT*PCMD" :command-vec [1 :v :w :x :y] :dir 1}
  :flat-trim {:command-class "AT*FTRIM"}
  :reset-watchdog {:command-class "AT*COMWDG"}
})
(def commands
  
  {:take-off        {:command-class "AT*REF" :command-bit-vec [9 18 20 22 24 28]}}
  :land            {:command-class "AT*REF" :command-bit-vec [18 20 22 24 28]}}
  :emergency       {:command-class "AT*REF" :command-bit-vec [8 18 20 22 24 28]}}
  :spin-right      {:command-class "AT*PCMD" :command-vec [1 0 0 0 :v] :dir 1}
  :spin-left       {:command-class "AT*PCMD" :command-vec [1 0 0 0 :v] :dir -1}
  :up              {:command-class "AT*PCMD" :command-vec [1 0 0 :v 0] :dir 1}
  :down            {:command-class "AT*PCMD" :command-vec [1 0 0 :v 0] :dir -1}
  :tilt-back       {:command-class "AT*PCMD" :command-vec [1 0 :v 0 0] :dir 1}
  :tilt-front      {:command-class "AT*PCMD" :command-vec [1 0 :v 0 0] :dir -1}
  :tilt-right      {:command-class "AT*PCMD" :command-vec [1 :v 0 0 0] :dir 1}
  :tilt-left       {:command-class "AT*PCMD" :command-vec [1 :v 0 0 0] :dir -1}
  :fly             {:command-class "AT*PCMD" :command-vec [1 :v :w :x :y] :dir 1}
  :flat-trim       {:command-class "AT*FTRIM"}
  :reset-watchdog  {:command-class "AT*COMWDG"}

)

(drone :take-off)
clj-drone

Show me the moves
Backup Video
just in case
Backup Video
just in case
Navigation Data
Navigation Data

@nav-data
All sorts of good streaming data

control-state: (flying, landed, hovering, etc...)
battery-percent
pitch
roll
yaw
altitude
velocity-x
velocity-y
velocity-z

Also vision tag (target) detection too.
All sorts of good streaming data

(set-log-data [:seq-num :battery-percent :control-state :detect-camera-type :targets-num :targets])

(drone-initialize)
(drone :init-targeting)
(drone :target-shell-h)
(drone :target-color-blue)
;; the drone will look for targets with blue tags on the horizontal camera
(drone :target-roundel-v)
;; the drone will look for the black and white roundel on the vertical camera
(drone-init-navdata)
Vision Processing

Video comes in as raw H264 format

Can save raw feed

Display raw feed

Conversion to PNG for more processing...
Facial Recognition!

OpenCV has Java Bindings
Something Missing
Beliefs and Goals
ASCRIBING MENTAL QUALITIES TO MACHINES

John McCarthy
Computer Science Department
Stanford University
Stanford, CA 94305
jmc@cs.stanford.edu
http://www-formal.stanford.edu/jmc/
1979

Abstract

Ascribing mental qualities like beliefs, intentions and wants to a machine is sometimes correct if done conservatively and is sometimes necessary to express what is known about its state. We propose some new definitional tools for this: definitions relative to an approximate theory and second order structural definitions.
Thermostat’s Beliefs & Goals
Thermostat’s Beliefs & Goals

The room is too cold
Thermostat’s Beliefs & Goals

The room is too cold
The room is too hot
Thermostat’s Beliefs & Goals

The room is too cold
The room is too hot
The room is just right
Thermostat’s Beliefs & Goals

The room is too cold
The room is too hot
The room is just right

**Goal:** The room should be just right
Why?

Easier to understand and reason
Why?

Easier to understand and reason

Useful concept for building intelligent systems
(def-belief-action ba-landed
  "I am landed"
  (fn [{:keys [control-state]}]
    (= control-state :landed))
  (fn [navdata] (drone :take-off)))
(def-belief-action ba-taking-off
  "I am taking off"
  (fn [{:keys [control-state]}]
    (= control-state :trans-takeoff))
  nil)
(def-goal g-take-off
  "I want to fly."
  (fn [{:keys [control-state]}]
    (= control-state :hovering))
  [ba-landed ba-taking-off])
(set-current-goal-list
 [g-take-off g-cruising-altitude g-land])
Flight with Beliefs and Goals Demo
Flight with Beliefs and Goals Demo

Backup video
Roomba

AR Drone
Roomba & Drone are Friends
Roomba & Drone are Friends

Backup Video ?
Recap

* Robots are great fun to program
Recap

* Robots are great fun to program

* Clojure is a powerful yet simple language – perfect for AI
Recap

* Robots are great fun to program
* Clojure is a powerful yet simple language – perfect for AI
* Ascribing beliefs and goals to machines can be useful
Recap

* Robots are great fun to program
* Clojure is a powerful yet simple language – perfect for AI
* Ascribing beliefs and goals to machines can be useful
* Robots communicating and acting together is the future
Resources

Github: gigasquid/clj-drone

Github: gigasquid/clj-roomba

Github: gigasquid/roomba-drone-friends

John McCarthy – Ascribing Mental Qualities to Machines

John McCarthy – All his papers
Credits

http://www.flickr.com/photos/midnightcomm/447335691/

http://good-wallpapers.com/cartoons/1934