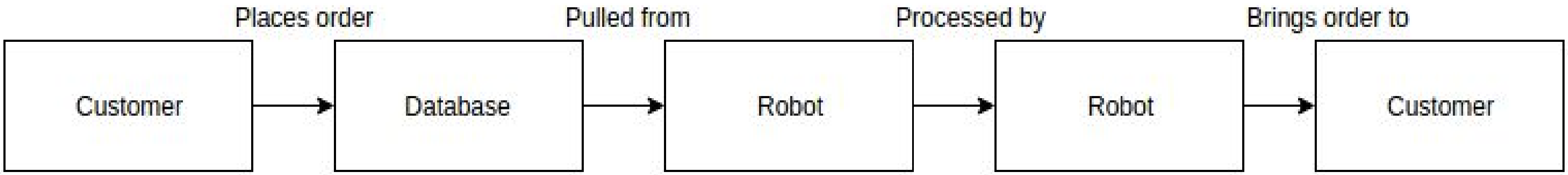
System Diagram



Places order

Pulled from

Processed by

Brings order to

Customer

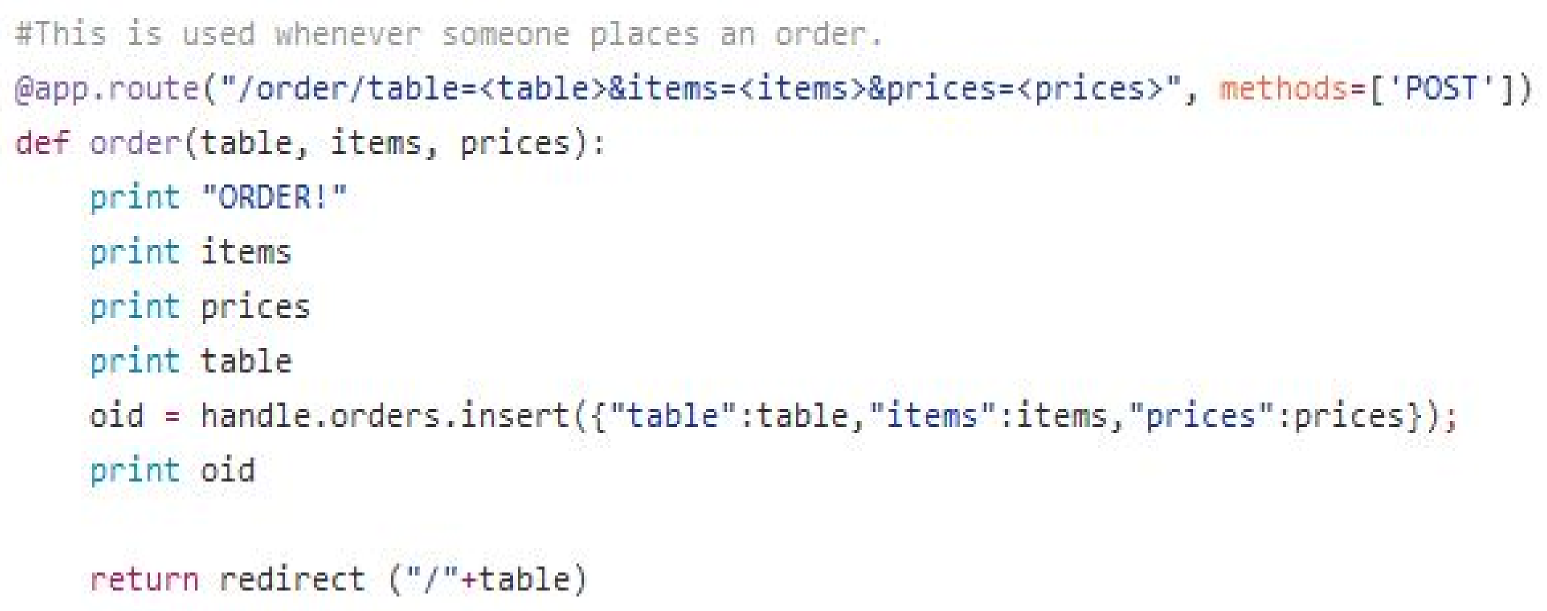
Database

Robot

Robot

Customer

Database Algorithms: Webapp CustomerUI



```
#This is used whenever someone places an order.
@app.route("/order/table=<table>&items=<items>&prices=<prices>", methods=['POST'])
def order(table, items, prices):
    print "ORDER!"
    print items
    print prices
    print table
    oid = handle.orders.insert({"table":table,"items":items,"prices":prices});
    print oid

    return redirect ("/"+table)
```

Database Algorithms: An Overview

- ROSE will pull orders FIFO order. For each order, ROSE parses the strings and stores the information into structs, then proceed to destroy the order in the database.
- On system startup, ROSE will initialize with documents pertaining to the status of ROSE. It will then proceed to periodically send information about its current status to the database.

DB Algorithms: Order Data Acquisition

```
void dbconn::recv_data(mongocxx::v_noabi::database db) {
    if (customer_order.parsed_items.size() != 0) {
        //then the order still exists, it has not been fulfilled
        return;
    }
    bsoncxx::document::element e;
    auto orders = db["orders"];
    auto cursor = orders.find(document{} << "items" << open_document << "$exists" << true
<< close_document << finalize);
    //acquire data from database
    for (auto&& doc : cursor) {
        e = doc["items"];
        customer_order.item = e.get_utf8().value.to_string();
        e = doc["prices"];
        customer_order.price = e.get_utf8().value.to_string();
        e = doc["table"];
        string table_string = e.get_utf8().value.to_string();
        customer_order.table = stoi(table_string);
        //delete the document
        orders.delete_one(doc);
        //break out the for loop so that we only remove one order
        break;
    }
    if (customer_order.item == "") {
        cout<<"the database is empty \n";
        return;
    }
    //store order into vector of vectors
    char* item_array = new char[customer_order.item.length() + 1];
    strcpy(item_array, customer_order.item.c_str());
    vector<string> init_item;
    customer_order.parsed_items.push_back(init_item);
    char* item_token = strtok(item_array,",");
```

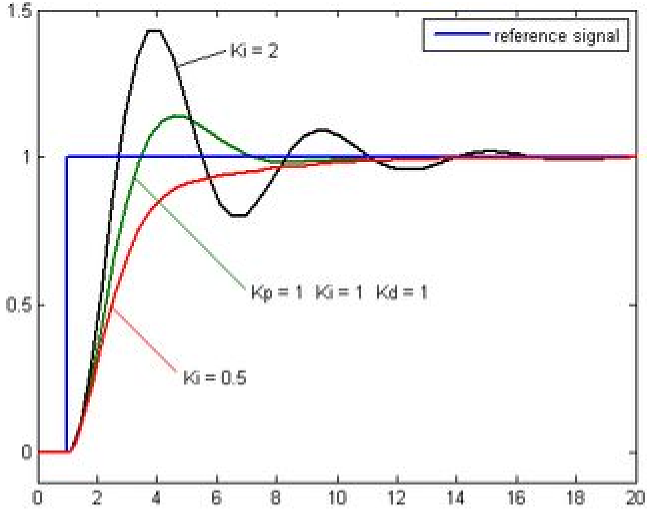
```
customer_order.parsed_items[0].push_back(string(item_token));
    //keep track of each item pushed to vector
    int count_item = 1;
    while (1) {
        //reinitialize for new item input
        vector<string> init_item;
        customer_order.parsed_items.push_back(init_item);
        item_token = strtok(NULL,",");
        if (item_token == NULL) {
            break;
        }
        customer_order.parsed_items[count_item].push_back(string(item_token));
        count_item++;
    }
    //now do same thing with "price_array"
    char* price_array = new char[customer_order.price.length() + 1];
    strcpy(price_array, customer_order.price.c_str());
    char* price_token = strtok(price_array,",");
    customer_order.parsed_items[0].push_back(string(price_token));
    //keep track of each item pushed to vector
    count_item = 1;
    while (1) {
        price_token = strtok(NULL,",");
        if (price_token == NULL) {
            break;
        }
        customer_order.parsed_items[count_item].push_back(string(price_token));
        count_item++;
    }

    delete item_array;
    delete price_array;
}
```

PID

Error correction algorithm

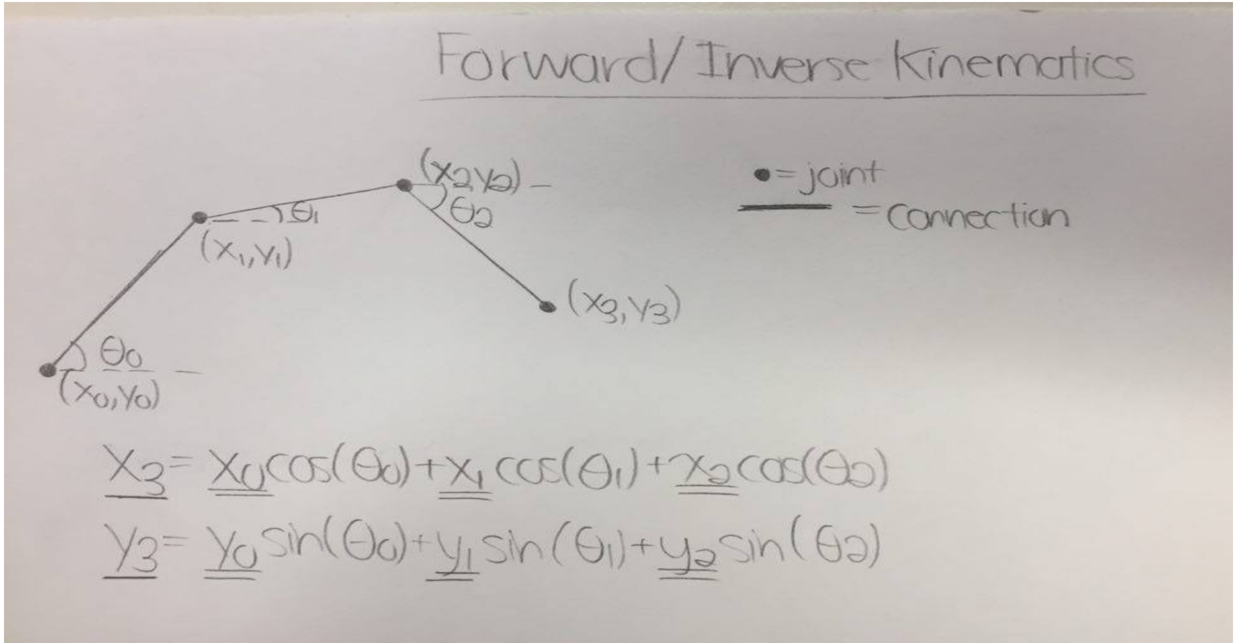
Incorporated in drive

$$u(t) = \overbrace{K_p e(t)}^{\text{Proportional}} + \overbrace{K_i \int_0^t e(\tau)\,d\tau}^{\text{Integral}} + \overbrace{K_d \frac{d}{dt} e(t)}^{\text{Derivative}}$$



Arm Control

- Forward Kinematics
- Inverse Kinematics
- Joint PID Control



Forward/Inverse Kinematics

$$\underline{X_3} = \underline{X_0}\cos(\theta_0) + \underline{X_1}\cos(\theta_1) + \underline{X_2}\cos(\theta_2)$$

$$\underline{Y_3} = \underline{y_0}\sin(\theta_0) + \underline{y_1}\sin(\theta_1) + \underline{y_2}\sin(\theta_2)$$

Localization

Particle Filter Localization

- Particles (Hypotheses)
- Weight Assignment
- Resample
- Particle Propagation
- Repeat

Computer Vision

→ Chilitags an OpenCV Library

1.Generates Chilitags

2. Gives you the 4 corners

→ Our Algorithm

1.Makes bounding box

2.Gives x,y,z positions

→ Chilitags Use Edge Detection

1.Points at which image brightness changes greatly or has any discontinuities

→ SVM (has a latency), Haar-Cascades(bad accuracy when training), Color Mapping(bad focus on correct object)