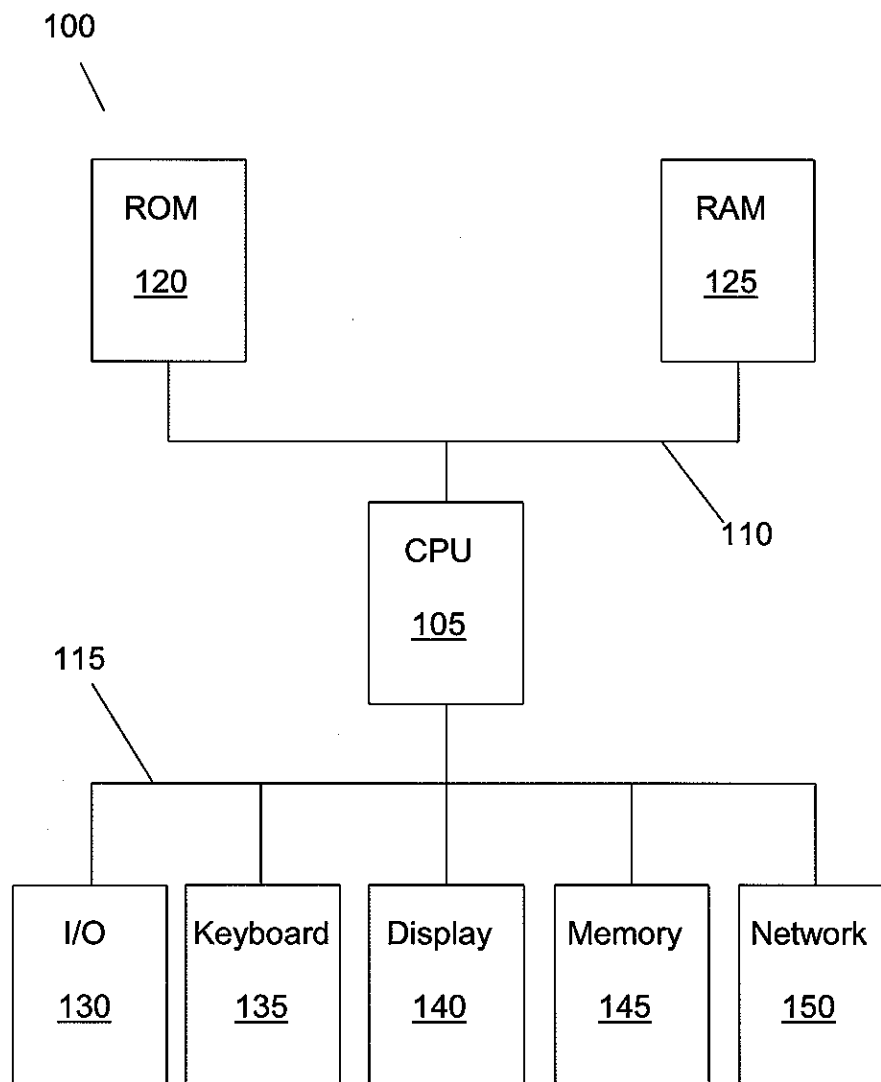
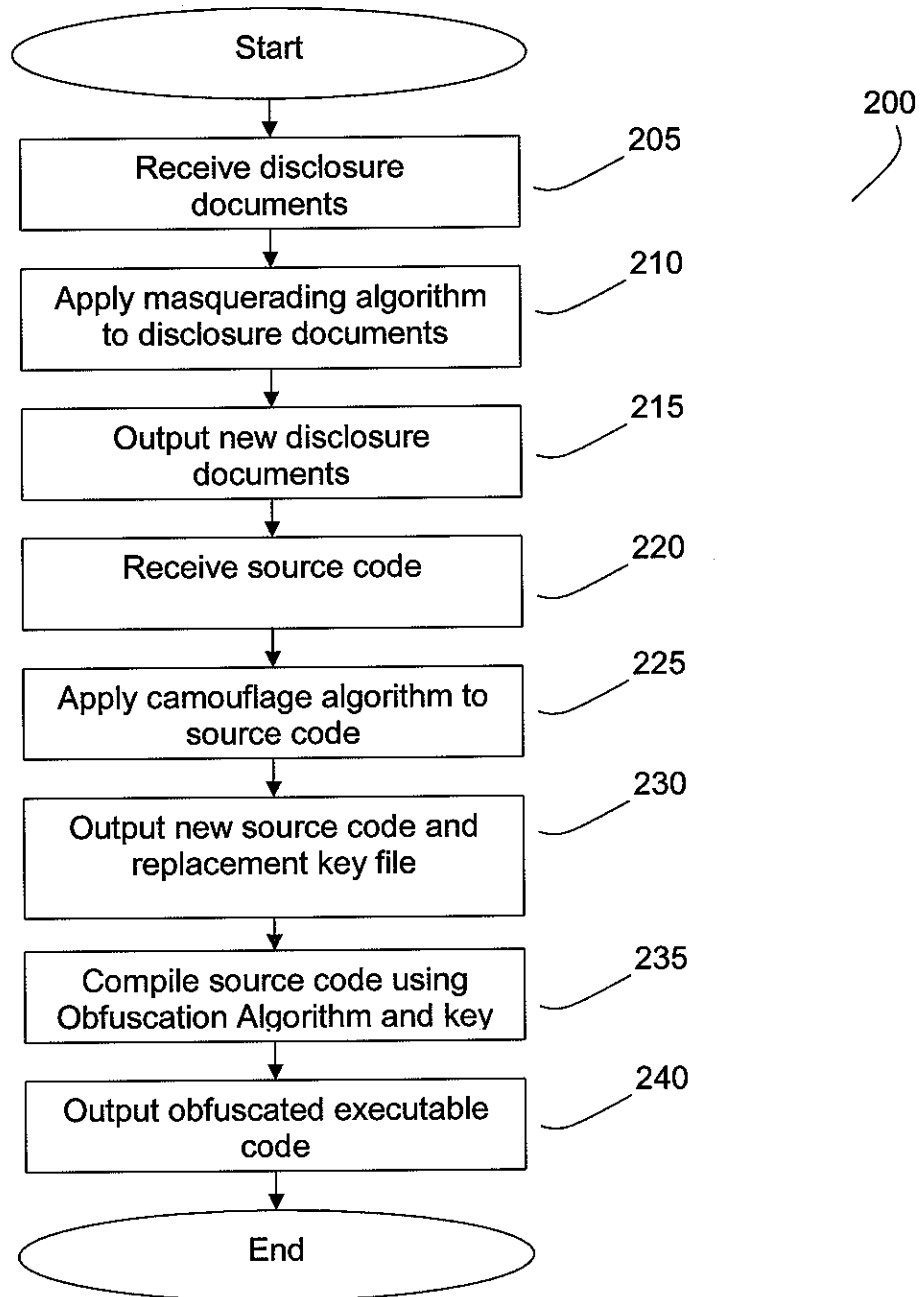


**Figure 1**

**Figure 2**

+ Name : ULTRA-Vis.txt
 + Description : text-based RFP file

300

Urban Leader Tactical Response, Awareness & Visualization (ULTRA-Vis)

305

BACKGROUND

Current military operations are focusing efforts on urban and asymmetric warfare, as well as distributed operations, but small unit leaders lack the capability to issue commands and share mission-relevant information in an urban environment non-line-of-sight. Various factors that can impact mission effectiveness and tempo of operations are:

leaders communicate by shouting and hand signals,
 teams operate within earshot and line-of-sight,
 intra-squad radios are hard to hear, and
 leaders must stop to use handheld displays.

Military operations in the urban terrain (extensive areas with hostile forces, non-combatant populations, and complex infrastructure) require special capabilities and agility to conduct close-combat operations under highly dynamic, adverse conditions. In short, tactical leaders need the ability to adapt on the move, coordinate small unit actions and execute commands across a wider area of engagement. Significant tactical advantages could be realized through the small unit leader's ability to intuitively generate/route commands and timely actionable combat information to the appropriate team or individual warfighter in a readily understood format that avoids information overload.

Thus, the goal of the ULTRA-Vis program is to enable the small unit commander to disseminate commands and actionable combat information over a wider area to support distributed, nonlinear, operations. To accomplish this goal, the ULTRA-Vis program will be focused on:

developing information processing techniques to recognize/interpret a small unit leader's gesture/ voice commands, develop techniques to create/disseminate/display geo-registered icons and actionable combat information for Fire Team Leaders/Dismount Warfighters in real time over an existing soldier radio network, and developing/integrating the necessary critical enabling technologies, such as 1) a low profile, see-through display, and 2) demonstrating multi-modal iconic C2 in a NLOS, urban environment with prototype systems.

310

PROGRAM OVERVIEW

ULTRA-Vis consists of critical technology components that provide the platoon or squad leader with the ability to issue commands and tactical information to the squad or individual fire teams (as depicted in the example shown in Figure 1 below). ULTRA-Vis will develop the key technologies that allow unit leaders and members to selectively transmit critical combat information in the form of icons using low bandwidth soldier voice and data radios.

The key technology areas to be developed include:

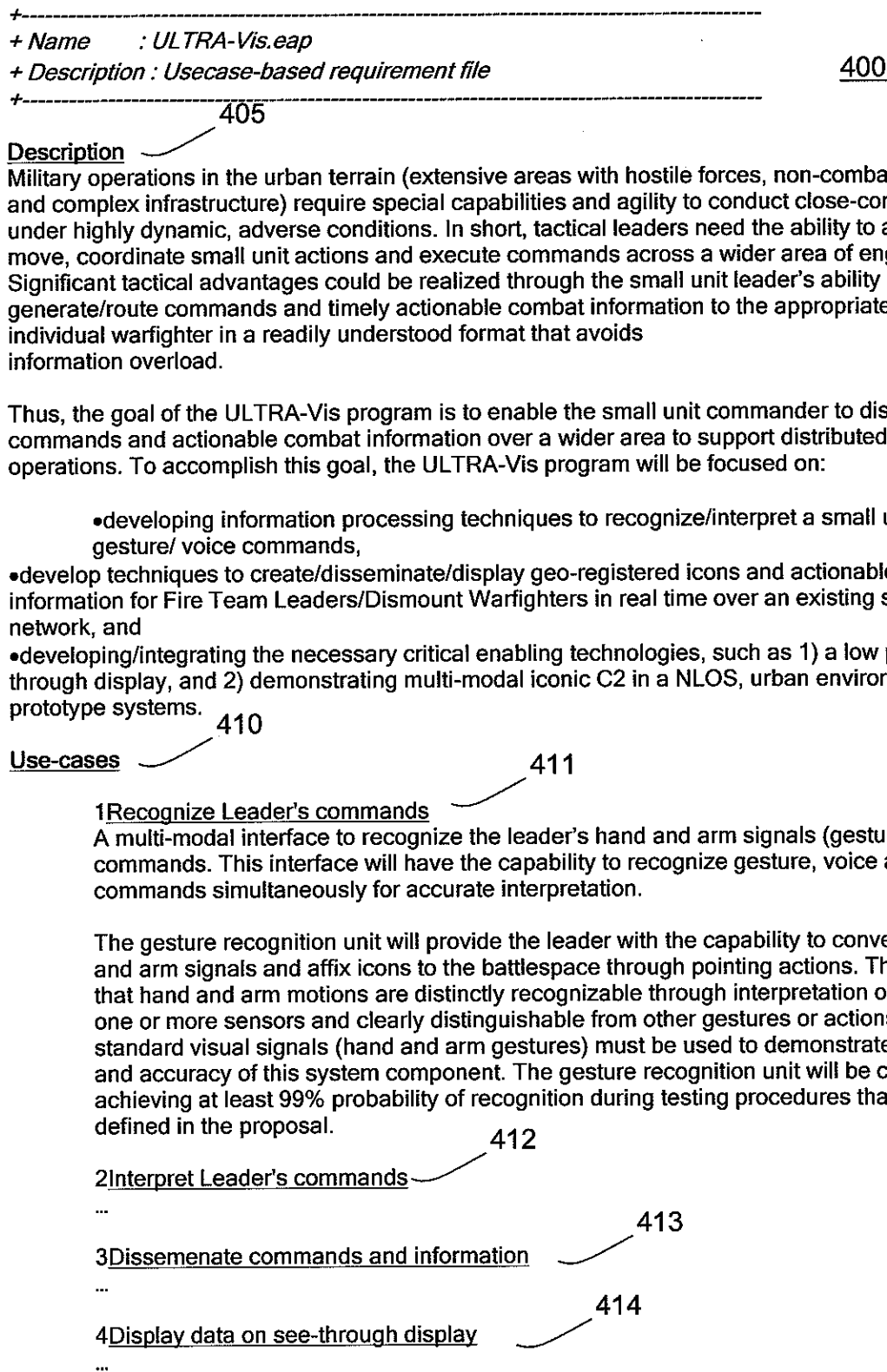
A multi-modal interface to recognize the leader's hand and arm signals (gestures) and voice commands. This interface will have the capability to recognize gesture, voice and tactile commands simultaneously for accurate interpretation.

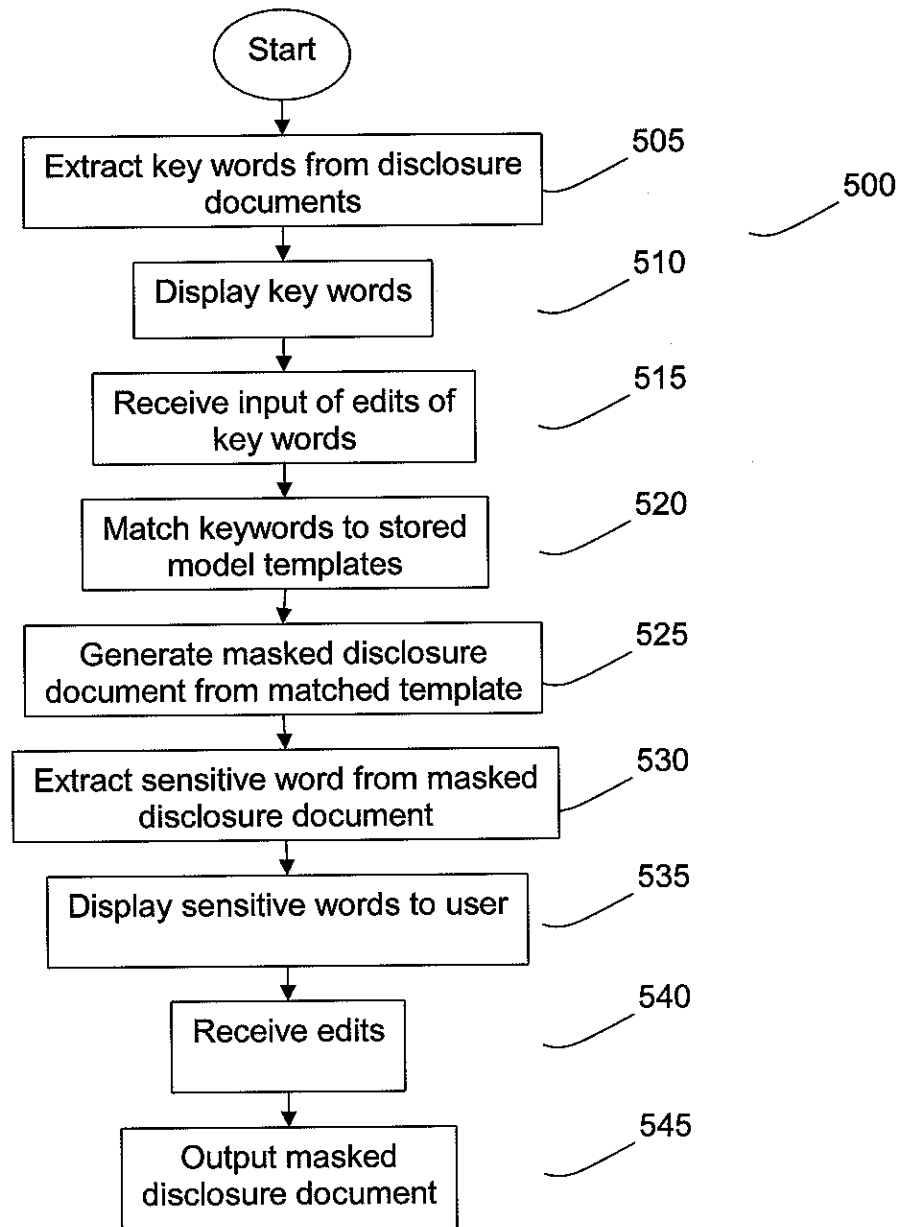
An information management engine to interpret the command and convert it to a standard, geo-registered icon or symbol for dissemination. The information management engine will not only convert the commands to standard iconic forms, but also will merge data from other contextual and off-board sources into the data packet to be transmitted to the team or individual intended to receive the command.

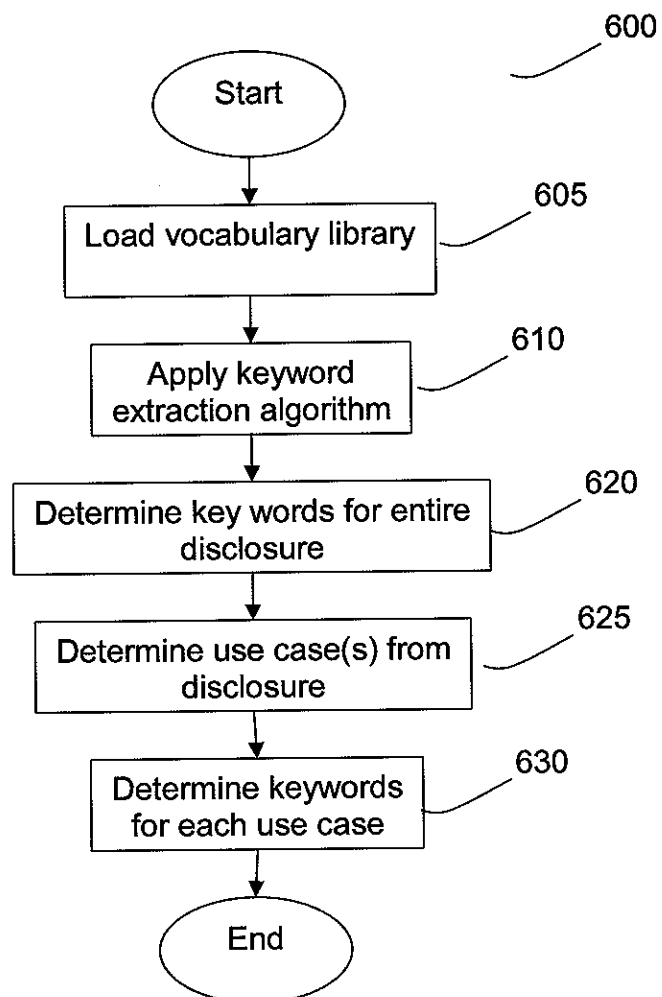
A see-through display to overlay the icon on the battlefield as seen from each warfighter's own perspective. The small unit will be outfitted with low- profile, non-occluding head-mounted displays for overlay of geo-registered icons on the real world environment. A head-tracking navigation unit will serve to align and display the icon from each warfighter's perspective.

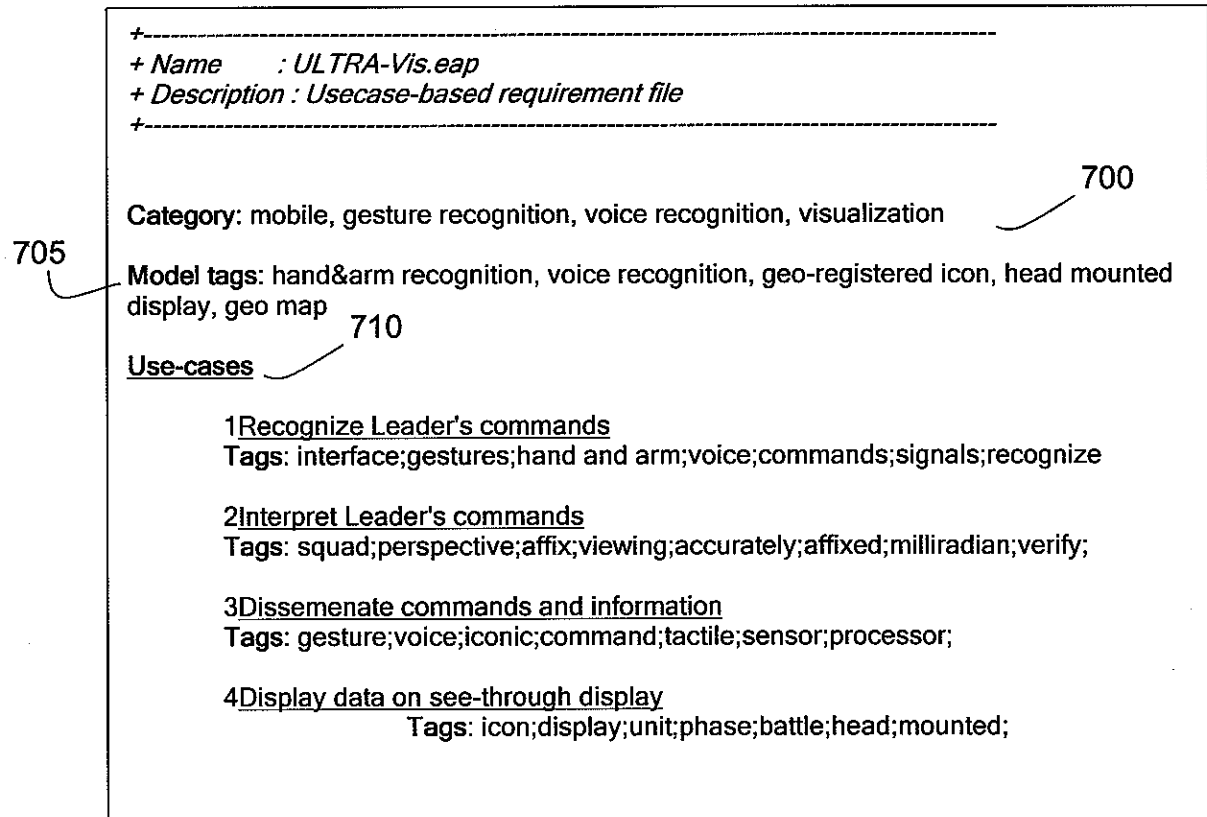
The advanced system will give each member of the unit the actionable combat information needed to increase awareness and enable dismounted units to be more effective in a distributed battle space. The system architecture shown below in Figure 2 is a notional concept describing functional capabilities and information flow for the ULTRA-Vis system. Offerors are encouraged to propose innovative approaches to achieve the highest level of overall system effectiveness, meeting or exceeding the program goals as outlined in this BAA. The proposed

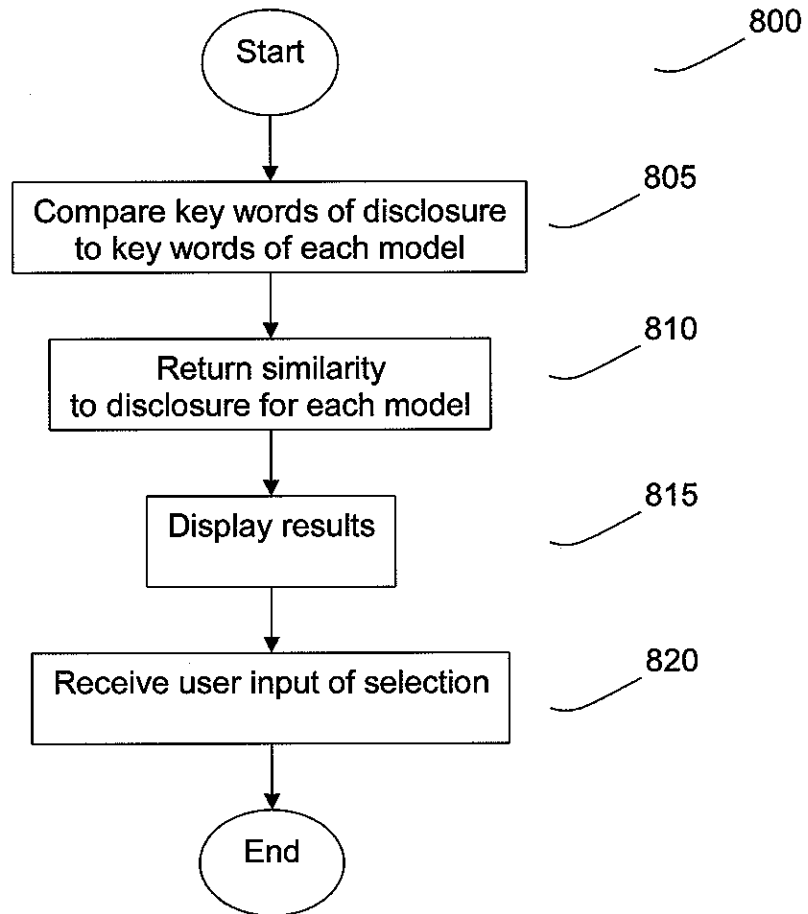
Figure 3

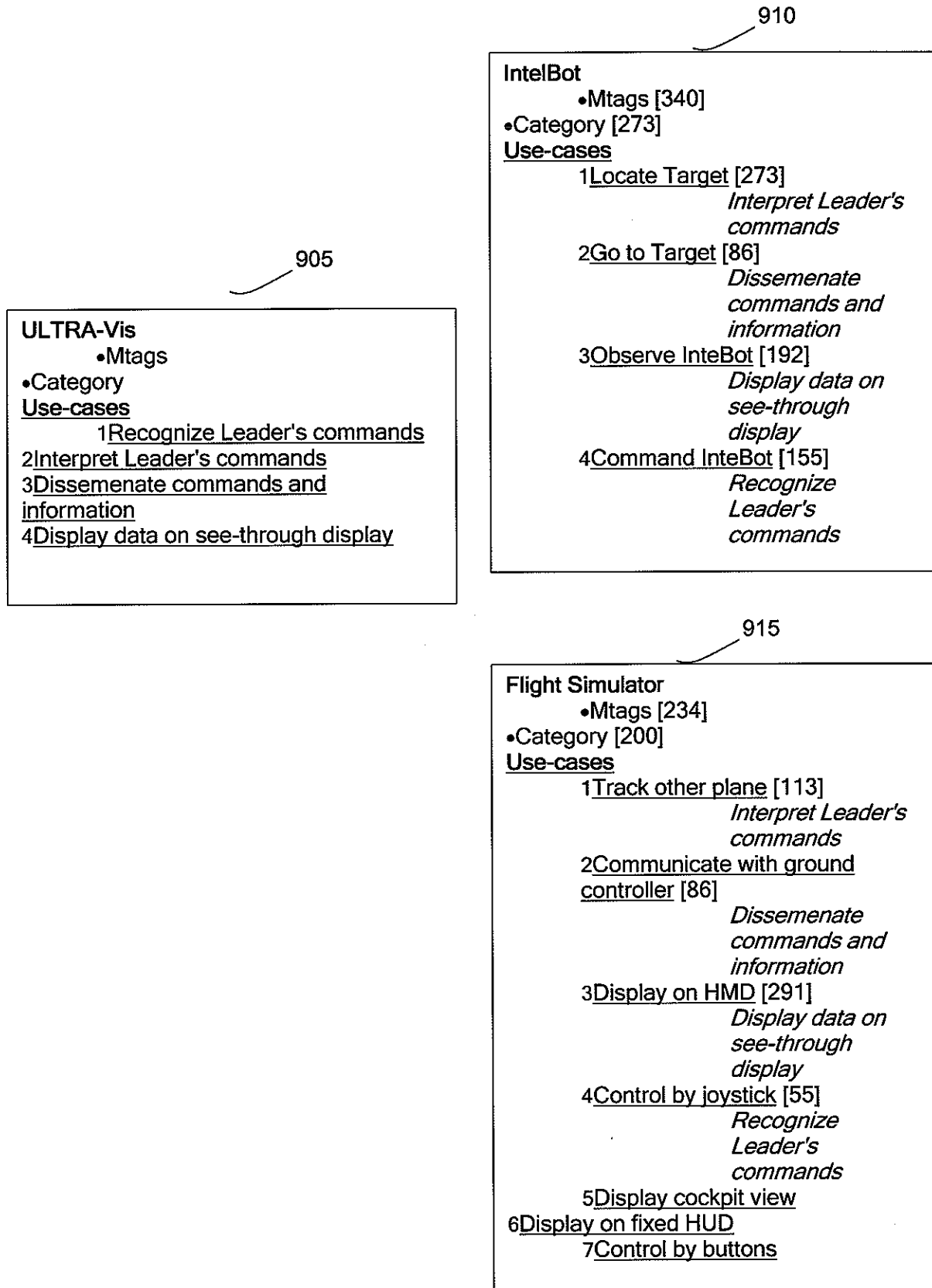
**Figure 4**

**Figure 5**

**Figure 6**

**Figure 7**

**Figure 8**

**Figure 9**

10/26

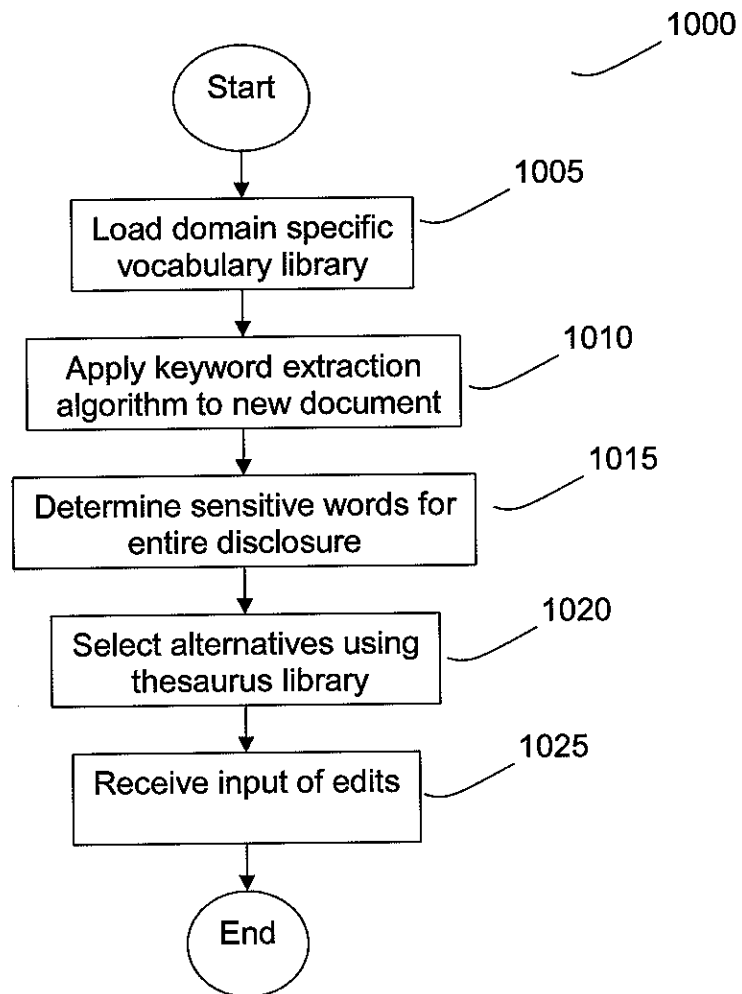


Figure 10

+-----+
 + Name : *ULTRA-Vis.eap*
 + Description : *Usecase-based requirement file*
 +-----+

Description

...

Use-cases

1Commands IntelBot

A multi-modal interface to recognize the leader's hand and arm signals (gestures) and voice commands. This interface will have the capability to recognize gesture, voice and tactile commands simultaneously for accurate interpretation.

The gesture recognition unit will provide the leader with the capability to convey standard hand and arm signals and affix icons to the battlespace through pointing actions. The task will show that hand and arm motions are distinctly recognizable through interpretation of data inputs from one or more sensors and clearly distinguishable from other gestures or actions. At least 10 standard visual signals (hand and arm gestures) must be used to demonstrate the functionality and accuracy of this system component. The gesture recognition unit will be capable of achieving at least 99% probability of recognition during testing procedures that will be clearly defined in the proposal.

2Interpret Operator's commands

...

3Communicate with IntelBot

...

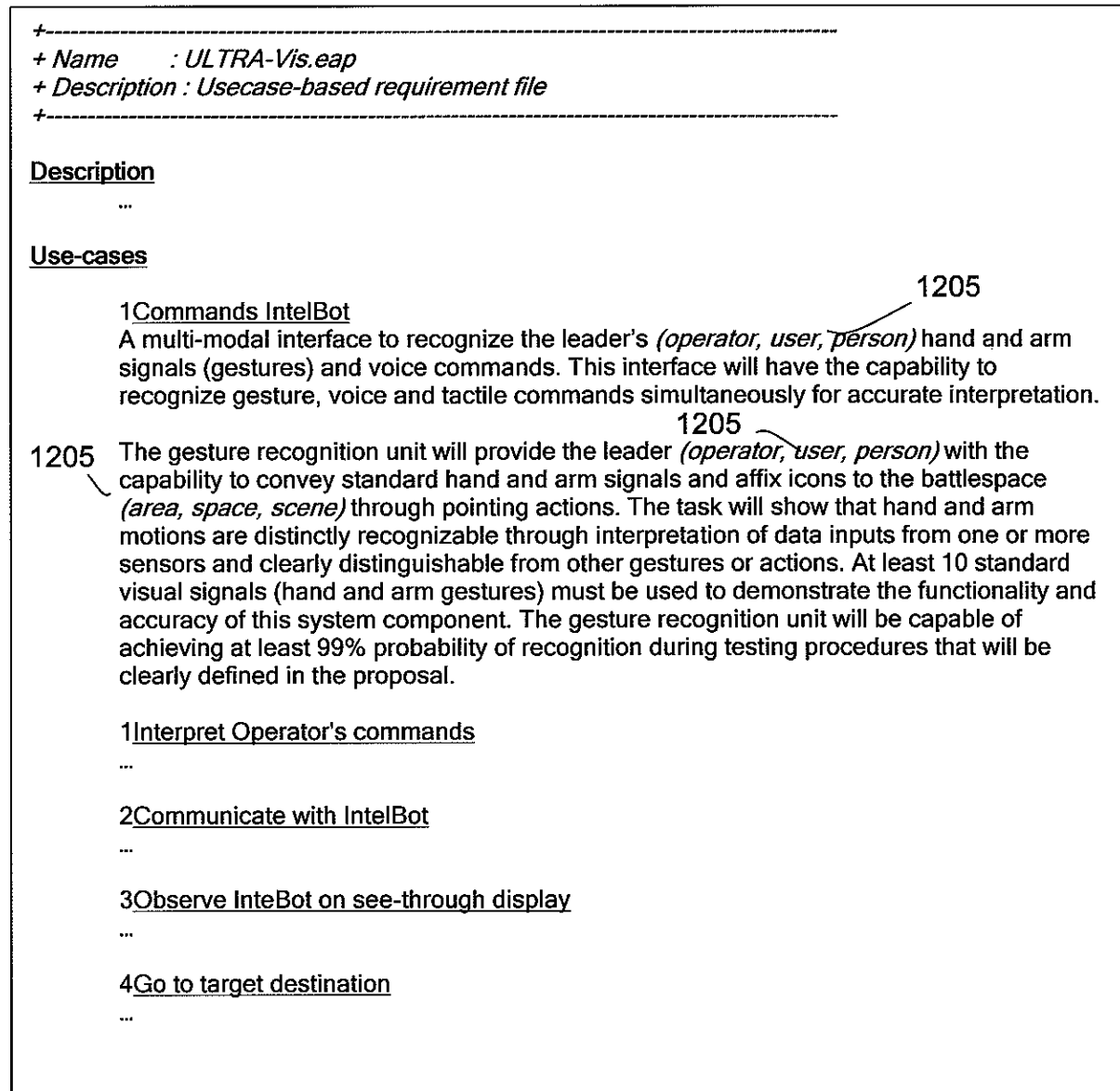
4Observe InteBot on see-through display

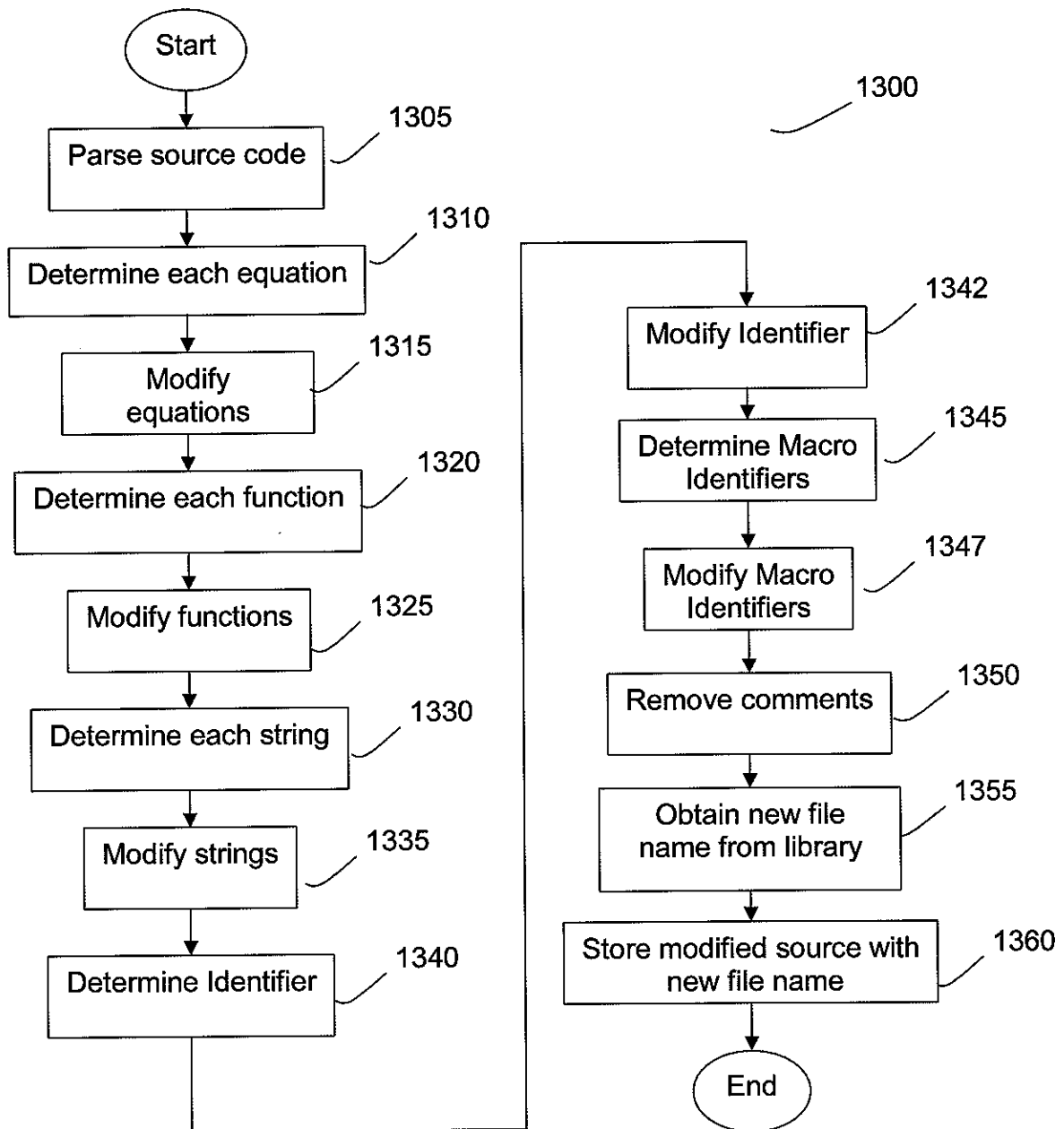
...

5Go to target destination

...

Figure 11

**Figure 12**

**Figure 13**

```

//=====
//
// Name      : myprogram.cpp
// Author    : C3S Pte Ltd
// Description : Foggy Camouflager sample
//=====
//
#include <iostream>
#include "compute.h"
#define MIN 200
#define MAX MIN + 100
#define AREA(radius) 3.14*3.14*radius

using namespace std;

int main() {
    char msg[] = "Danger, reduce speed";
    Computation center = new Computationi();
    Vehicle v1 = new Vehicle();
    Vehicle v2 = new Vehicle();
    cout << "First Vehicle area: " << AREA(v1.R) << endl;
    printf("Second Vehicle area: %f\n", AREA(v2.R));
    float speed = center.velocity(time, v1.x, v1.y);
    printf("Speed of vehicle 1: %f\n", speed);
    if (center.collide(v1, v2))
        cout << msg;
    return 0;
}

```

1405

```

// Name      : compute.h
//=====
//
Class Computation {
    public float velocity(int time, float x_coordinate, float y_coordinate);
    public bool collide(vehicle v1, vehicle v2);
}

```

1410

```

// Name      : compute.cpp
//=====
//
#ifndef COMPUTATION_CC
#define COMPUTATION_CC

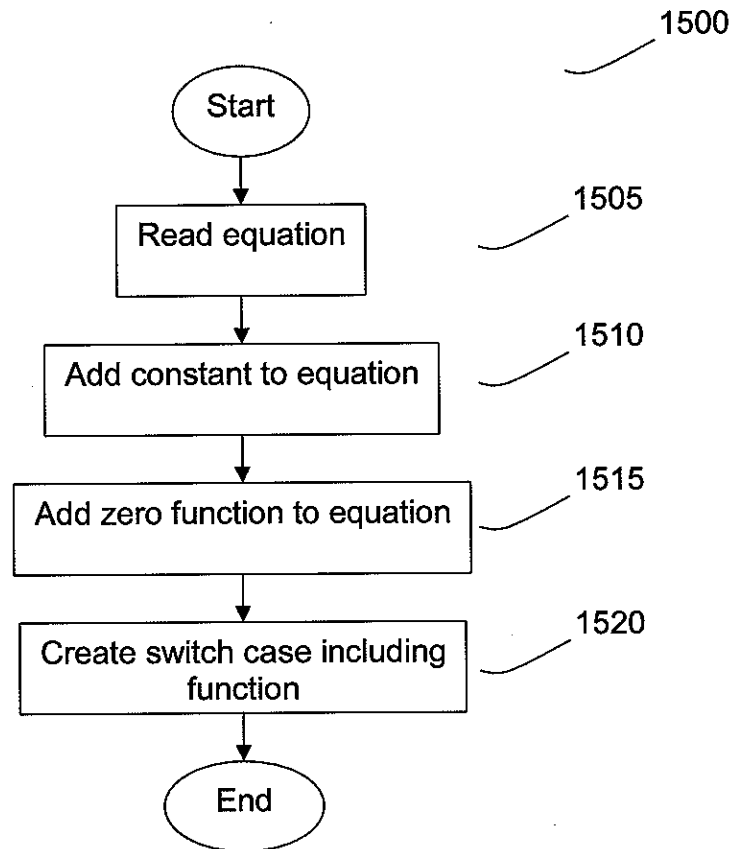
float Computation::velocity(int time, float x_coordinate, float y_coordinate) {
    float v = 0;
    v = (time*time - 4.5*time + 1.76) * x_coordinate / (time*time*time/2 + y_coordinate);
    return v;
}

bool Computation::collide(Vehicle v1, Vehicle v2) {
    // Distance between 2 centers
    float distance = 0;
    distance = (v1.x - v2.x)*(v1.x - v2.x) + (v1.y - v2.y)*(v1.y - v2.y);
    if (distance < (v1.R + v2.R)*(v1.R + v2.R))
        return true;
    else
        return false;
}

```

1415

Figure 14

**Figure 15**

```

// Name      : compute.cpp
//=====================================================
===
#ifndef COMPUTATION_CC
#define COMPUTATION_CC

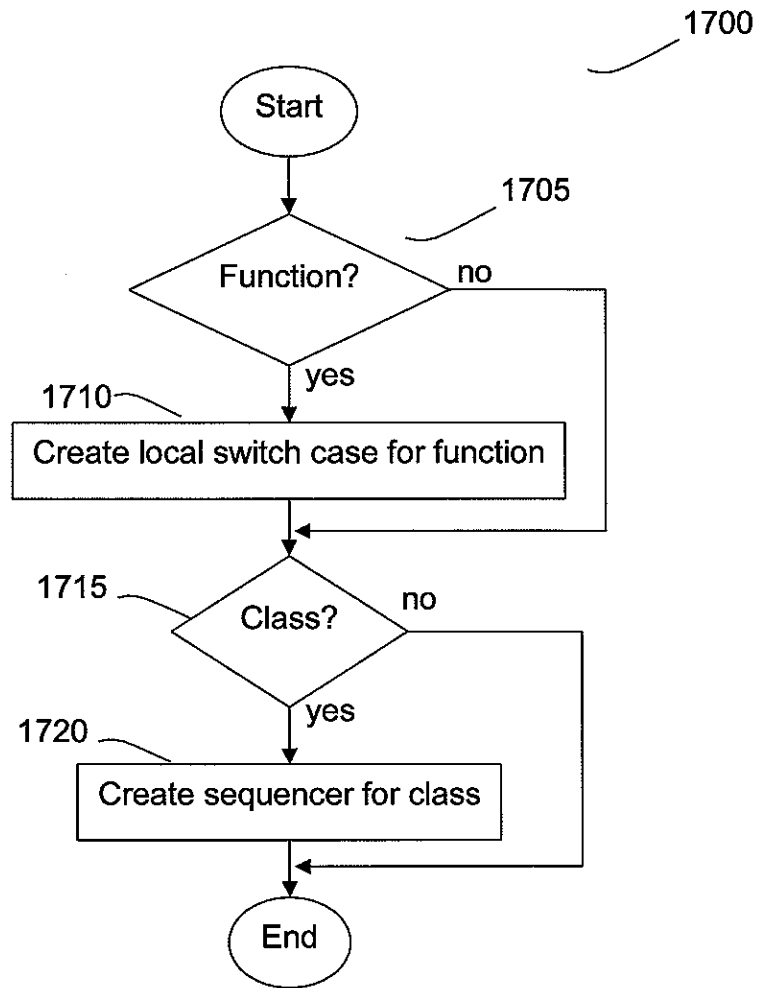
float Computation::velocity(int time, float x_coordinate, float y_coordinate) {
    float v = 0;
    v = (time*time - CAM::getNumber(493)*time + CAM::getNumber(478)) * x_coordinate /
        (time*time*time/ CAM::getNumber(313) + y_coordinate);
    return v;
}

bool Computation::collide(Vehicle v1, Vehicle v2) {
    // Distance between 2 centers
    float distance = 0;
    switch (CAM::getNumber(17)) {
        case 372:
            distance = (v1.x + v2.x)*(v1.x - v2.x) - (v1.y - v2.y)*(v1.y - v2.y) + CAM::getNumber(3);
            break;
        case 4012:
            distance = (v1.x - v2.x)*(v1.x - v2.x) + (v1.y - v2.y)*(v1.y - v2.y) + CAM::getNumber(98);
            break;
    }

    if (distance < sin(2*v1.R)+cos(2*v1.R)+2*sin(v1.R)*sin(v1.R)-
        (cos(v1.R)+sin(v1.R))*(cos(v1.R)+sin(v1.R))+v1.R + v2.R)*(v1.R + v2.R))
        return true
    else
        return false;
}
#endif

```

Figure 16

**Figure 17**

```
// Name      : compute.cpp
//=====
=====
#ifndef COMPUTATION_CC
#define COMPUTATION_CC

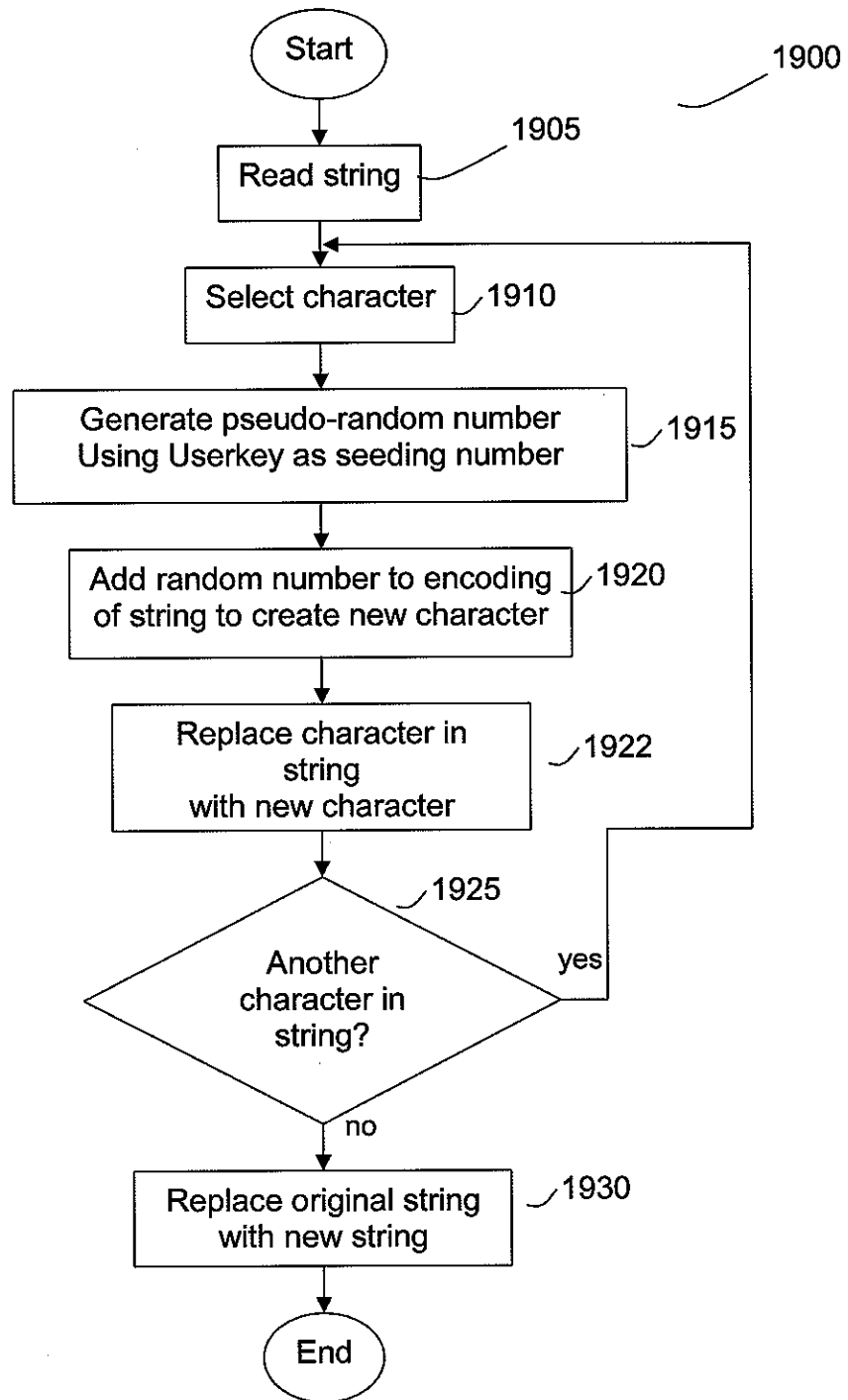
void* Computation::sequencer(int ID, int p1, vehicle p2, float p3, vehicle p4, float p5) {
    switch (CAM::getNumber(ID)){
        case 479:    return &velocity(p1, p3, p5);
        case 123:    return &velocity(p1, p4, p3);
        case 093:    return &collide(p2, p4);
        case 472:    return &collide(p4, p2);
    }
}
```

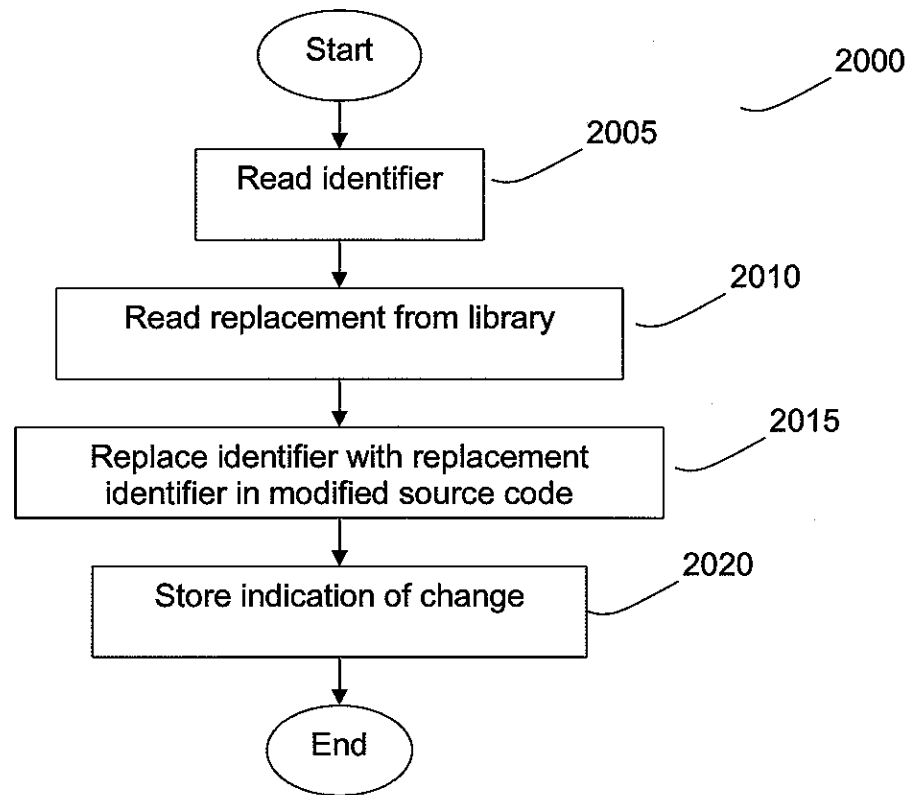
1805

```
...
using namespace std;

int main() {
    char msg[] = "Danger, reduce speed";
    Computation center = new Computationi();
    Vehicle v1 = new Vehicle();
    Vehicle v1 = new Vehicle();
    cout << "First Vehicle area: " << AREA(v1.R) << endl;
    printf("Second Vehicle area: %f\n", AREA(v1.R));
    float speed;
    switch (CAM::getNumber(23)){
        case 470:
            speed = (float)*(center.sequencer(748, time, v1, v1.x, v2, v1.y));
            break;
        case 702:
            speed = (float)*(center.sequencer(748, time, v2, v2.x, v2, v1.y));
            break;
    }
    printf("Speed of vehicle 1: %f\n", speed);
    if ((bool)*(center.sequencer(736, time, v1, v1.x, v2, v1.y)))
        cout << msg;
    return 0;
}
```

1810**Figure 18**

**Figure 19**

**Figure 20**

```

...
using namespace std;

int main() {
    char msg[] = "508~535~536~456~535~526~456~507~540~521~523~531~482~456~";
    Computation center = new Computationi();
    Vehicle v1 = new Vehicle();
    Vehicle v1 = new Vehicle();
    cout<<CAM::getString("~1226~1259~1240~1242~1250~1175~1248~1258~1175~1244~1
252~1255~1259~1264~1201~1175~", 606)<< AREA(v1.R) << endl;
    printf(CAM::getString("~449~480~481~401~466~479~401~470~477~470~478~470~479
~485~427~401~%fn" , 915), AREA(v1.R);
    float speed;
    switch (CAM::getNumber(23)){
        case 470:
            speed = (float)*(center.sequencer(748, time, v1, v1.x, v2, v1.y));
            break;
        case 702:
            speed = (float)*(center.sequencer(748, time, v2, v2.x, v2, v1.y));
            break;
        }
    printf("387~408~421~412~416~408~423~408~421~365~339~%fn",speed);
    if ((bool)*(center.sequencer(736, time, v1, v1.x, v2, v1.y)))
        cout << msg;
    return 0;
}

```

Figure 21

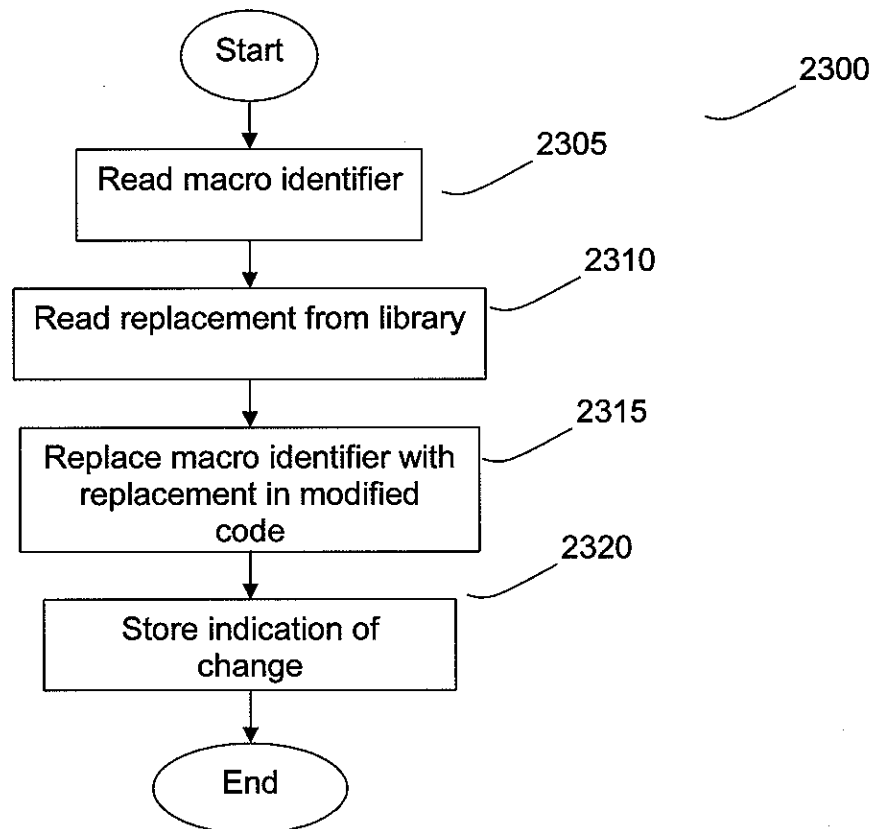
```

...
using namespace std;

int main() {
    char pigeon[] = "508~535~536~456~535~526~456~507~540~521~523~531~482~456~";
    Eagle crow = new Eagle();
    Penguin v1 = new Penguin();
    Penguin v1 = new Penguin();
    cout<<BIRD::chicken("~1226~1259~1240~1242~1250~1175~1248~1258~1175~1244~125
2~1255~1259~1264~1201~1175~", 606)<< AREA(v1.R) << endl;
    printf(BIRD::chicken("~449~480~481~401~466~479~401~470~477~470~478~470~479~48
5~427~401~%fn", 915), AREA(v1.R);
    float hawk;
    switch (BIRD::duck(23)){
        case 470:
            hawk = (float)*( crow.crane(748, time, v1, v1.x, v2, v1.y);
            break;
        case 702:
            hawk = (float)*( crow. crane(748, time, v2, v2.x, v2, v1.y);
            break;
        }
    printf("387~408~421~412~416~408~423~408~421~365~339~%fn",speed);
    if ((bool)*(crow. crane(736, time, v1, v1.x, v2, v1.y)))
        cout << pigeon;
    return 0;
}

```

Figure 22

**Figure 23**

```

/*27*/
#include <iostream>
#include "henrry.h"
#define LEMON 200
#define ORANGE LEMON + 100
#define APPLE(pear) 3.14*3.14*pear

using namespace std;

int main() {
    char pigeon[] = "508~535~536~456~535~526~456~507~540~521~523~531~482~456~";
    Eagle crow = new Eagle();
    Penguin v1 = new Penguin();
    Penguin v1 = new Penguin();
    cout<<BIRD::chicken("~1226~1259~1240~1242~1250~1175~1248~1258~1175~1244~12
52~1255~1259~1264~1201~1175~", 606)<< APPLE(v1.R) << endl;
    printf(BIRD::chicken("~449~480~481~401~466~479~401~470~477~470~478~470~479~
485~427~401~%fn", 915), APPLE (v1.R);
    float hawk;
    switch (BIRD::duck(23)){
        case 470:
            hawk = (float)*( crow.crane(748, time, v1, v1.x, v2, v1.y);
            break;
        case 702:
            hawk = (float)*( crow. crane(748, time, v2, v2.x, v2, v1.y);
            break;
    }
    printf("387~408~421~412~416~408~423~408~421~365~339~%fn",speed);
    if ((bool)*(crow. crane(736, time, v1, v1.x, v2, v1.y)))
        cout << pigeon;
    return 0;
}

```

Figure 24

```

/*27*/
#include <iostream>
#include "henry.h"
#define LEMON 200
#define ORANGE LEMON + 100
#define APPLE(pear) 3.14*3.14*pear

using namespace std;

int main() {
    char pigeon[] =
"508~535~536~456~535~526~456~507~540~521~523~531~482~456~";
    Eagle crow = new Eagle();
    Penguin v1 = new Penguin();
    Penguin v1 = new Penguin();
    cout<<BIRD::chicken("1226~1259~1240~1242~1250~1175~1248~1258~1175~1244
~1252~1255~1259~1264~1201~1175~", 606)<< APPLE(v1.R) << endl;
    printf(BIRD::chicken("449~480~481~401~466~479~401~470~477~470~478~470~4
79~485~427~401~%fn", 915), APPLE (v1.R);
    float hawk;
    switch (BIRD::duck(23)){
        case 470:
            hawk = (float)*( crow.crane(748, cuckoo, v1, v1.x, v2, v1.y);
            break;
        case 702:
            hawk = (float)*( crow. crane(748, cuckoo, v2, v2.x, v2, v1.y);
            break;
    }
    printf("387~408~421~412~416~408~423~408~421~365~339~%fn",speed);
    if ((bool)*(crow. crane(736, time, v1, v1.x, v2, v1.y)))
        cout << pigeon;
    return 0;
}

```

```

/*23*/
Class Eagle {
    public void* crane(int, int, Penguin, float, Penguin, float)
    public float flamingo(int cuckoo, float seagull, float swallow);
    public bool swan(Penguin v1, Penguin);
}

```

Figure 25

```

/*12*/
#ifndef CAMUCAMU
#define CAMUCAMU

void* Eagle::crane(int ID, int p1, vehicle p2, float p3, vehicle p4, float p5) {
    switch (BIRD::duck(ID)){
        case 479:    return &flamingo(p1, p3, p5);
        case 123:    return &flamingo(p1, p4, p3);
        case 093:    return &swan(p2, p4);
        case 472:    return &swan(p4, p2);
    }
}

float Eagle::flamingo (int cuckoo, float seagull, float swallow) {
float v = 0;
    v = (cuckoo * cuckoo - BIRD::duck(493)*cuckoo + BIRD::duck(478)) * seagull / (cuckoo *
    cuckoo * cuckoo / BIRD::duck(313) + swallow);
    return v;
}

bool Eagle::swan(Penguin v1, Penguin v2) {
    /*22*/
    float owl = 0;
    switch (BIRD::duck(17)) {
        case 372:
            owl = (v1.x + v2.x)*(v1.x - v2.x) - (v1.y - v2.y)*(v1.y - v2.y) + BIRD::duck(3);
            break;
        case 4012:
            owl = (v1.x - v2.x)*(v1.x - v2.x) + (v1.y - v2.y)*(v1.y - v2.y) + BIRD::duck(98);
            break;
    }

    if (owl < sin(2*v1.R)+cos(2*v1.R)+2*sin(v1.R)*sin(v1.R)-
    (cos(v1.R)+sin(v1.R))*( cos(v1.R)+sin(v1.R))+(v1.R + v2.R)*( v1.R + v2.R))
        return true
    else
        return false;
}
#endif

```

Figure 26