BAB II

LANDASAN TEORI

2.1. Tahap Perkembangan Motorik Anak

Motorik merupakan perkembangan pengendalian gerakan tubuh melalui kegiatan yang terkoordinir antara susunan saraf, otot, otak, dan *spinal cord*. Perkembangan motorik meliputi motorik kasar dan halus. Motorik kasar adalah gerakan tubuh yang menggunakan otot-otot besar atau sebagian besar atau seluruh anggota tubuh yang dipengaruhi oleh kematangan anak itu sendiri. Contohnya kemampuan duduk, menendang, berlari, naik-turun tangga dan sebagainya.

Sedangkan motorik halus adalah gerakan yang menggunakan otot-otot halus atau sebagian anggota tubuh tertentu, yang dipengaruhi oleh kesempatan untuk belajar dan berlatih. Misalnya, kemampuan memindahkan benda dari tangan, mencoret-coret, menyusun balok, menggunting, menulis dan sebagainya. Kedua kemampuan tersebut sangat penting agar anak bisa berkembang dengan optimal. Perkembangan motorik sangat dipengaruhi oleh organ otak. Otak lah yang mensetir setiap gerakan yang dilakukan anak.Semakin matangnya perkembangan sistem syaraf otak yang mengatur otot memungkinkan berkembangnya kompetensi atau kemampuan motorik anak. Perkembangan motorik anak dibagi menjadi dua:

1. Keterampilan atau gerakan kasar seperti berjalan, berlari, melompat, naik turun tangga.

2. Keterampilan motorik halus atau keterampilan manipulasi seperti menulis, menggambar, memotong, melempar dan menagkap bola serta memainkan benda-benda atau alat-alat mainan [2].

Berikut adalah tabel beberapa tahap perkembangan anak [3]:

Tabel 2.1 Tahap Perkembangan Anak Usia 1-5 Tahun

No	USIA ANAK	MOTORIK KASAR	MOTORIK HALUS
1	1-2 TAHUN	1. Merangkak 2. Berdiri dan berjalan beberapa langkah 3. Berjalan cepat 4. Cepat-cepat duduk agar tidak jatuh 5. Merangkak di tangga 6. Berdiri di kursi tanpa pegangan 7. Menarik dan mendorong benda-benda berat 8. Melempar bola	1. Mengambil benda kecil dengan ibu jari atau telunjuk 2. Membuka 2-3 halaman buku secara bersamaan 3. Menyusun menara dari balok 4. Memindahkan air dari gelas ke gelas lain 5. Belajar memakai kaus kaki sendiri 6. Menyalakan TV dan bermain remote 7. Belajar mengupas pisang
2	2-3 TAHUN	1. Melompat-lompat 2. Berjalan mundur dan jinjit 3. Menendang bola 4. Memanjat meja atau tempat tidur 5. Naik tangga dan lompat di anak tangga terakhir 6. Berdiri dengan 1 kaki	1. Mencoret-coret dengan 1 tangan 2. Menggambar garis tak beraturan 3. Memegang pensil 4. Belajar menggunting 5. Mengancingkan baju 6. Memakai baju sendiri
3	3-4 TAHUN	1. Melompat dengan 1 kaki 2. Berjalan menyusuri papan 3. Menangkap bola besar 4. Mengendarai sepeda 5. Berdiri dengan 1 kaki	1. Menggambar manusia 2. Mencuci tangan sendiri 3. Membentuk benda dari plastisin 4. Membuat garis lurus dan lingkaran cukup rapi
4	4-5 TAHUN	1. Menuruni tangga dengan cepat	1. Menggunting dengan cukup baik

		2. Seimbang saat berjalan mundur 3. Melompati rintangan 4. Melempar dan menangkap bola 5. Melambungkan bola	2. Melipat amplop 3. Membawa gelas tanpa menumpahkan isinya 4. Memasukkan benang ke lubang besar

2.2. Permainan

Untuk melatih otot motorik anak, pelatihan harus dilakukan dengan kondisi yang menyenangkan, maka dari itu untuk melatih otot motorik anak dapat dilakukan dengan bermain. Permainan adalah kegiatan terstruktur, biasanya dilakukan untuk kesenangan dan terkadang digunakan sebagai alat pendidikan. Permainan berbeda dari pekerjaan, yang biasanya dilakukan untuk remunerasi, dan berbeda dari seni, yang lebih peduli dengan ekspresi ide. Perbedaan tersebut tidak jelas, dan banyak permainan juga dianggap bekerja atau seni.

Komponen kunci dari permainan adalah tujuan, aturan, tantangan, dan interaksi. Permainan biasanya melibatkan stimulasi mental atau fisik, dan terkadang kedua-duanya. Banyak permainan sebagai bentuk latihan, atau melakukan peran pendidikan, simulasional atau psikologis.

Menurut perancangan permainan computer Chris Crawford, definisi istilah permainan menggunakan serangkaian dikotomi:

1. Ekspresi kreatis adalah seni jika dibuat untuk kecantikan sendiri, dan hiburan jika dibuat untuk uang.
2. Hiburan adalah sebuah mainan yang jika dapat melakukan interaksi. Film dan buku-buku dianggap sebagai contoh hiburan non-interaktif.

3. Jika tidak ada tujuan yang terkait dengan permainan, itu adalah mainan. Jika memiliki tujuan, permainan adalah tantangan.
4. Jika tantangan tidak memiliki "*active agent against whom you compete*" maka disebut teka-teki, jika ada itu adalah konflik.
5. Terakhir, jika para pemain hanya dapat mengalahkan lawan, tetapi tidak menyerang mereka, mengganggu kinerja mereka, konflik sebuah kompetisi. Namun, jika serangan diizinkan, kemudian konflik memenuhi syarat sebagai permainan.

Permainan dapat dicirikan oleh "apa pemain dapat lakukan". Hal ini sering disebut *gameplay*. Elemen kunci umum diidentifikasi dalam konteks ini adalah alat dan aturan yang mendefinisikan konteks keseluruhan permainan dan yang pada gilirannya menghasilkan keterampilan, strategi, dan kesempatan.

Permainan sering diklasifikasikan oleh komponen yang diperlukan untuk memainkannya (misalnya miniature, bola, kartu, papan *and pieces* atau komputer). Permainan seperti petak kumpet memanfaatkan alat apapun, melainkan aktivitas mereka ditentukan oleh lingkungan.

Selain yang sering dikarakteristik dengan alat-alat yang digunakan, permainan sering ditentukan oleh aturan yang digunakan. Ketika aturan yang dikenakan variasi dan perubahan, perubahan cukup dalam aturan biasanya menghasilkan permainan baru. Aturan umunya menentukan urutan gilirannya, hak-hak dan tanggung jawab pemain dan tujuan masing-masing pemain.

Kondisi menang yang umum adalah menjadi yang pertama untuk mengumpulkan kuota tertentu, poin atau token (seperti dalam *settlers of catan*),

yang memiliki jumlah terbesar token pada akhir permainan (seperti dalam monopoli), atau beberaa hubungan token permainan seseorang untuk orang lawan seseorang (seperti dalam catur atau *checkmate*).

Ketika permainan menggunakan alat dan aturan, maka muncul kebutuhan akan keterampilan, strategi, keberuntungan atau kombinasi dari semuanya.

2.3. Jenis-jenis Permainan Berbasis Teknologi

Tahun 1952, AS Douglas membuat *game* Tic tac toe yang ditampilkan dalam sebuah tabung vakum computer. Kemudian tahun 1958, Willy Higginbotham membuat *game* Tennis for Two yang berjalan di osiloskop yang terhubung ke analog *Donner computer*. Tahun 1961 – 1962, *game space war* dikembangkan di MIT menggunakan grafik vector di PDP-1. Sega merilis periskop, terciptalah elektronik *shooting game*, *game arcade* yang pertama.

Tahun 1971, Nolan Bushnell mengembangkan *computer space*, *arcade game* komersil yang pertama. Berbasis pada *spacewar*. Menggunakan grafik vector, *real time* dan terlalu *sophisticated* untuk pasar. Dan gagal.

Tahun 1972, Bushnell memulai Atari. Odyssey oleh Magnavox membuat *game* hockey. Home TV *game* yang pertama, masih analog. Terjual 100.00 dengan harga jual $100/console. Tahun 1973, Atari membuat *Pong in Arcades*. Tahun 1974, Kee merilis *Tank*. Atari membuat *game* balap yang pertama. Tahun 1972-1976, William Crowther dan Don Woods membuat *game adventure* : *The colossal cave*. *Game* petualangan berbasis text yang pertama, berjalan di DEC *mainframes* (PDP - 10) dilewat dan akhirnya tahun 2004, The sims 2, halo 2, half

life 2, doom. *Game console*, pertumbuhannya stabil, harga lebih murah. Nokia meluncurkan N-Gages, Nintendo meluncurkan DS, Sony meluncurkan PSP [16].

Ada beberapa jenis *game* yang saat ini berkembang diantaranya:

1. ***FPS (First Person Shooter)***

   Adalah jenis *game* tembak-tembakan dengan tampilan pemain adalah tokoh yang dimainkan. Biasanya berupa misi untuk suatu tujuan tertentu. Ciri khas *game* ini adalah penggunaan senjata jarak jauh. Contoh : *Counter-Strike, Call of Duty, Ghost Recon, Point Blank, Crossfire*

2. ***RTS (Real Time Strategy)***

   Adalah genre permainan strategi, taktik, dan logika. Biasanya berupa perang yang tiap pemainnya memiliki suatu pasukan atau negara. Dalam RTS permainan dapat berupa sejarah, fantasi, dan fiksi ilmiah. Contoh : *Age of Empires, Warcraft, Command & Concuer, Rise of Nations, Stronghold.*

3. ***RPG (Role Playing Game)***

   Adalah *game* yang para pemainnya memainkan peran tokoh-tokoh khayalan dan berkolaborasi untuk merajut sebuah cerita bersama. Ada pula jenis lain *game* RPG ini, diantaranya :

   a. LARP (*Live Action Role Playing*) adalah *game* RPG dimana para pemain bisa melakukan gerakan fisik tokohnya. Biasanya pemain menggunakan kostum dan menggunakan alat-alat yang sesuai dengan tokoh, dunia dan cerita yang dia mainkan.

b. MMORPG (*Massively Multiplayer Online Role-Playing Game*) adalah *game* RPG yang melibatkan ribuan pemain untuk bermain *game* bersama dalam dunia maya.

Contoh : Final Fantasy, Ragnarok, Avalon, RF, World of Warcraft, DotA, Perfect World.

4. *Construction and Management Simulation Games*

Adalah *game* yang mensimulasikan proyek membangun dan mengelola. Pada dasarnya adalah masalah ekonomi dan konseptual. *Game* ini jarang yang melibatkan konflik dan eksplorasi, dan hampir tidak pernah meliputi tantangan fisik. Contoh *: Sim City, Roller Coster Tycoon, Caesar*

5. *Vehicle Simulation*

Jenis permainan ini mensimulasi pengoperasian beberapa kendaraan, kendaraan bisa berupa pesawat terbang, pesawat tempur, kereta, kendaraan perang, maupun kendaraan konstruksi. Contoh : *Train Simulator, Truck Simulator, FlightGear, Tram, Orbiter.*

6. *Adventure Games*

Game yang mengutamakan masalah eksplorasi dan pemecahan teka-teki. Namun terkadang meliputi masalah konseptual, dan tantangan fisik namun sangat jarang.Contoh *: Indiana Jones, God of War, Tomb Raider, Assasins Creed* [16].

2.4. *Collision Detection*

Collision detection adalah komponen penting pada game. disadari atau tidak, komponen ini sering kali membuat game menjadi realistik, dan terkadang membuat pemain marah jika *collision detection* tidak berjalan dengan sesungguhnya. *Collision detection* adalah fungsi yang dapat mendeteksi tabrakan antara 2 obyek atau lebih.

Collision detection ini juga berguna untuk menentukan posisi dari satu obyek dengan obyek yang lain sehingga tidak ada obyek yang saling menembus. Sehingga game yang akan dibuat memiliki kesamaan dengan realitas yang ada.

Collision detection memiliki teknik yg berbeda untuk diterapkan pada pembuatan game 2D dan game 3D.

Teknik *Collision detection* pada 2D ada 2 cara yaitu :

a. *Rectangles Collision detection*

Teknik ini merupakan pendeteksian tabrakan menggunakan *bound* berbentuk persegi, dimana setiap *sprite* atau obyek yang dibuat dalam game memiliki *bound* masing-masing. Cara ini cukup mudah karena hanya perlu membandingkan apakah ada *intersection*(irisan) pada kedua obyek pada koordinat tertentu. Namun cara ini kurang akurat karena untuk *sprite* atau obyek yang memiliki ruang kosong yang besar dengan bound. Hal ini akan muncul saat kondisi kedua batas obyek tersebut beririsan walaupun sebenarnya tidak terjadi tabrakan maka akan dianggap terjadi tabrakan. Cara ini cocok untuk *game* yang tidak memperhatikan akurasi deteksi tabrakan

atau untuk *game* yang menggunakan sprite yang bentuknya mendekati persegi.

Adapun cara lain dari hasil modifikasi teknik ini yaitu *reduced size bounding box*. Dimana kotak batasnya diperkecil dengan mengurangi beberapa pixel dari ukuran *sprite* atau obyek yang sebenarnya. Sehingga dengan cara ini pada saat melakukan *collision detection* akan lebih akurat karena ruang kosong antara *sprite* dan *bound* berkurang.

Kemudian cara lain dari modifikasi teknik ini adalah dengan cara menggunakan beberapa *bounding box* secara bersamaan untuk satu *sprite* atau obyek. Contohnya untuk *sprite* berbentuk manusia, pertama *bounding box* digunakan untuk bagian kepala kemudian *bounding box* yang lainnya digunakan untuk badan dan kaki. Cara ini lebih akurat karena banyaknya *bounding box* yang digunakan dapat disesuaikan dengan bentuk gambar pada *sprite*. Namun semakin banyak *bounding box* yang digunakan akan semakin banyak pula waktu yang digunakan untuk *collision detection* [19] .

b. *Circle Rectangle Collision Detection*

Untuk teknik ini memiliki cara yang hampir sama dengan yang sebelumnya hanya saja disini *bound* atau batas yang digunakan berbentuk lingkaran. Cara pengecekannya lebih mudah karena linggkaran memiliki jarak yg sama ke batas luarnya. Caranya yaitu dengan menghitung jarak kedua obyek dan membandingkannya dengan jumlah radius masing-masing *bound* dari kedua obyek, jika jaraknya lebih kecil dari jumlah radius kedua obyek maka terjadi tabrakan(*collision*) dan jika lebih besar maka tidak

terjadi tabrakan. Cara sebelumnya yaitu *rectangle collision detection* lebih akurat daripada cara ini.

Teknik C*ollision detection* pada 3D ada 2 cara yaitu :

a. *Bounding Box Collision Detection*

Teknik ini menggunakan cara yang hampir sama dengan teknik *Rectangle Collision Detection* pada 2D hanya batas kotaknya berbentuk 3D uang melingkupi setiap obyek. Untuk melakukan pengecekan apakah kotak-kotak itu saling beririsan atau tidak kita cukup membandingkan nilai maksimum dan minimum pada koordinat x,y dan z.

b. *Sphere Collision Detection*

Untuk teknik ini *bound* atau batas yang digunakan berbentuk bola yang melingkupi setiap obyek. *collisions detection* yang dilakukan sebagai berikut :

Obyek A berpusat di ax,ay,az dan memiliki radius ar

Obyek B berpusat di bx,by,bz dan memiliki radius br

Irisan terjadi jika :

(ax-bx)2+(ay-by)2+(az-bz)2 < (ar+br)2

(2.1)[19].

2.5. *Augmented reality* (AR)

Augmented reality (AR), atau dikenal dengan realitas tertambah adalah teknologi yang menggabungkan benda maya dua dimensi dan atau tiga dimensi kedalam sebuah lingkuangan nyata tiga dimensi lalu memproyeksikan benda-benda maya tersebut dalam waktu nyata.

Benda-benda nyata menampilkan informasi yang tidak dapat diterima oleh pengguna dalam inderanya sendiri. Hal ini membuat realitas tertambah sesuai sebagai alat untuk membantu persepsi dan interaksi penggunanya dengan dunia nyata. Informasi yang ditampilkan oleh benda maya membantu pengguna melaksanakan kegiatan-kegiatan dalam dunia nyata. AR dapat diaplikasikan untuk semua indera, termasuk pendengaran, sentuhan, dan penciuman. Selain digunakan dalam bidang-bidang seperti kesehatan, militer, industry manufaktur, realitas tertambah juga telah diaplikasikan dalam perangkat-perangkat yang digunakan orang banyak, seperti pada telepon genggam.

Ada dua definisi dari AR yang diterima secara umum, salah satunya definisi dari Ronald Azuma pada tahun 1997, Azuma mendifinisikan bahwa:

1. Menggabungkan kenyataan dan virtual
2. Interaktif secara *real time*
3. Tergolong kedalam lingkungan 3D

Selain itu, Paul Milgram dan Fumio Kishino mendifinisikan *Milgram's Reality-Virtual Continuum* pada tahun 1994. Mereka mendeskripsikan sebuah kontinum yang membentang dari lingkungan nyata ke lingkungan virtual murni. Diantaranya terdapat *augmented reality* (lebih dekat ke lingkungan nyata) dan *augmented virtuality* (lebih dekat ke lingkungan virtual). Diagram mengenai *Milgram's Reality-Virtuality Continuum* dapat dilihat pada gambar 2.1

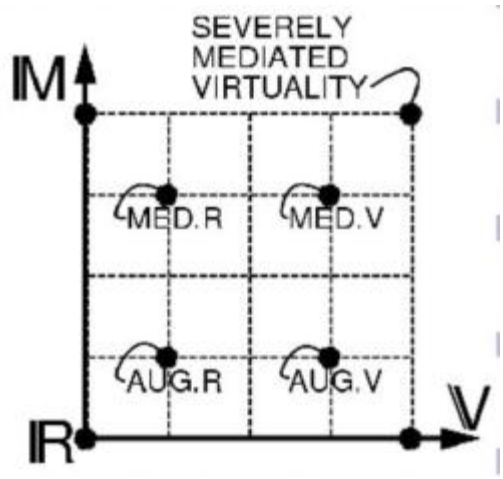


Gambar 2.1 Diagram mengenai *Milgram's Reality-Virtuality Continuum*

AR sejatinya variasi lain dari realitas virtual. Teknologi realitas virtual membenamkan pengguna secara total pada lingkungan sintetis, ketika masuk dalam dunia buatan itu, kita tidak dapat mengenali lingkungan nyata disekitanya. Namun AR tidak memisahkan yang nyata dengan virtual, yang terjadi adalah penggabungan antara keduanya pada ruang yang sama. Selain menambahkan benda maya dalam lingkungan nyata, realitas tertambah juga berpotensi menghilangkan banda-benda yang sudah ada. Menambah sebuah lapisan gambar maya dimungkinkan untuk menghilangkan atau menyembunyikan lingkungan nyata dari pandangan pengguna. Misalnya, untuk menyembunyikan sebuah meja dalam lingkungna nyata, perlu digambarkan lapisan representasi tembok dan lantai kosong yang diletakkan diatas gambar meja nyata, sehingga menutupi meja nyata dari pandangan pengguna.

Kontinum ini diperluas lagi menjadi bidang dua dimensi dari *virtuality* dan *mediality*. Titik asal R menunjukan kenyataan yang tidak dimodofikasi. Sebuah kontinum di sumbu *virtuality* V mencakup realitas tertambah dangan grafis (*augmented reality*), serta grafis tertambah dengan realitas (*augmented virtuality*). Namun taksonomi ini juga mencakup modifikasi realitas atau kombinasi dari semuanya. Modifikasi ini dilambangkan dengan bergerak naik ke sumbu

mediality. Selanjutnya dari sumbu ini, dapat ditemukan *mediated reality*, *mediated virtuality*, atau kombinasi dari semuanya. Lebih lanjut ke atas dan kekanan terdapat dunia virtual yang responsive ke versi realitas yang sangat termodifikasi. Diagram mengenai *mediated reality continuum* dapat dilihat pada gambar 2.2



Gambar 2.2 ***Mediated Reality Continuum***

2.6. *Tracking Marker*

Dalam proses perancangan dan implementasi aplikasi permainan ini menggunakan metode *tracking marker* yaitu proses deteksi keberadaan penanda atau marker yang berbentuk matriks. Untuk perancangan marker dibuat tidak monoton seperti biasanya yang hanya menggunakan gambar hitam persegi. Marker yang digunakan memiliki bentuk gambar yang rumit dan penuh warna sehingga bentuk dari marker lebih interaktif. Semakin rumit gambar yang digunakan untuk menjadi sebuah marker maka proses *tracking marker* akan berjalan lebih baik.

Salah satu dukungan dari *library* IN2AR adalah proses *tracking marker* dapat menggunakan gambar yang lebih rumit sehingga permainan yang akan

dirancang terlihat lebih menyenangkan. Gambar yang akan digunakan sebagai *marker* perlu kita konversi terlebih dahulu menjadi file berekstensi *.ass sebelum dapat digunakan dalam kode. Sayangnya hingga saat ini proses konversi masih perlu melibatkan IN2AR *support team.*[9] Walaupun setelah dikonversi file gambar yang berekstensi *.ass dapat diubah ke ekstensi lain misalnya (*.islam). Contoh gambar yang dapat dijadikan sebagai marker adalah uang 100 ribu rupiah, seperti pada gambar 2.3



Gambar 2.3 Contoh Gambar Rumit Digunakan sebagai Marker

Cara kerja dari pencarian *marker* ini dilakukan secara per-*frame* pada video secara *real-time*. Ketika *marker* ditemukan, akan dilakukan kalkulasi *world transformation matrix* berdasarkan posisi dan kemiringan dari marker. Selanjutnya hasil dari *world transformation matrix* posisi obyek virtual terhadap obyek fisik marker dapat ditentukan. dibutuhkan *smoothing* agar hasil *marker tracking* menjadi lebih halus.

Namun dalam melakukan *tracking marker* IN2AR memiliki beberapa keterbatasan, adapun keterbatasan tersebut adalah sebagai berikut:

1. **Oklusi**

   Ketika sebagian besar marker tertutup oleh benda, maka marker tidak adak dapat terdeteksi, salah satu hal penyebab kegagalan deteksi adalah hampir setengah gambar marker tertutupi oleh benda.

2. **Pencahayaan**

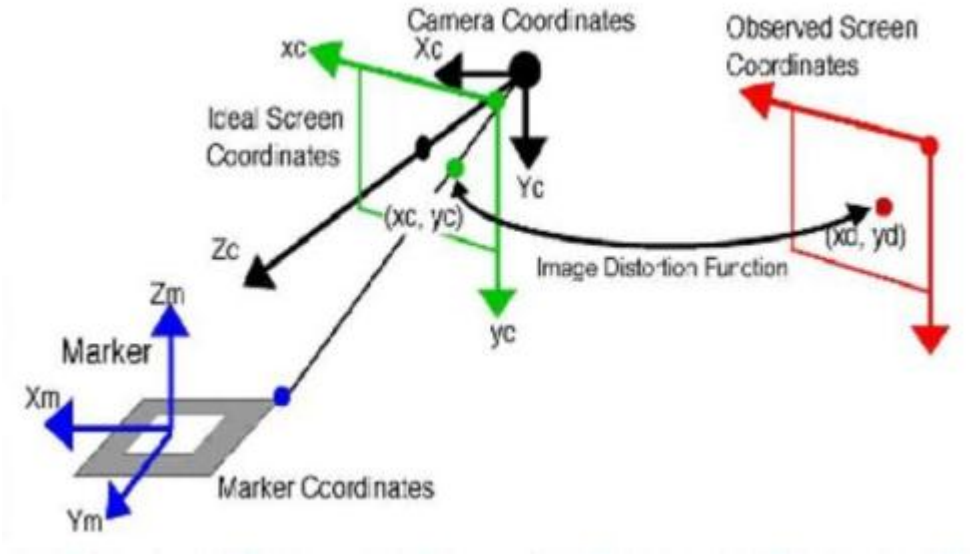
   Dalam proses pendeteksian marker sangat tergantung pada pencahayaan yang digunakan. Marker akan sulit dideteksi jika pencahayaan yang digunakan terlalu gelap membuat gambar sulit terlihat atau pencahayaan yang digunakan terlalu terang sehingga gambar memantulkan cahaya yang menyilaukan. Akan tetapi jika menggunakan marker dengan warna yang lebih rumit sangat memudahkan dalam proses pendeteksian.

3. **Jarak**

   Jarak yang terlalu jauh menyebabkan pola marker yang ditangkap oleh kamera menjadi buram dan tidak jelas sehingga marker dideteksi.

4. **Perangkat Kamera**

   Perangkat kamera yang buruk menyebabkan gambar yang ditangkap menjadi kurang berkualitas. Misalnya saja karena resolusi atau focus kamera yang buruk, gambar yang ditangkap menjadi buram.



Gambar 2.4 Penentuan Transformasi Matrix dari Marker

2.7. IN2AR

IN2AR merupakan sebuah *engine* selain FLARToolKit untuk Adobe Flash Player. Dengan kemampuan natural feature recognition yang dimiliki IN2AR dapat menggunakan gambar apapun sebagai marker. Jika pada FLARToolkit semakin sederhana dan resolusi marker semakin kecil maka akan semakin memudahkan dalam *tracking marker*, maka justru sebaliknya pada IN2AR, semakin rumit dan detail gambar yang digunakan sebagai marker maka semakin baik dalam proses *tracking marker* yang akan dilakukan.

IN2AR juga menyediakan *free license* yang memperbolehkan pengguna untuk mengkomersilkan aplikasi yang dibuat menggunakan IN2AR dengan keterbatasan hanya pada kewajiban mencantumkan logo IN2AR. Adapun keterbatasan lain dalam menggunakan *engine* ini adalah gambar yang akan

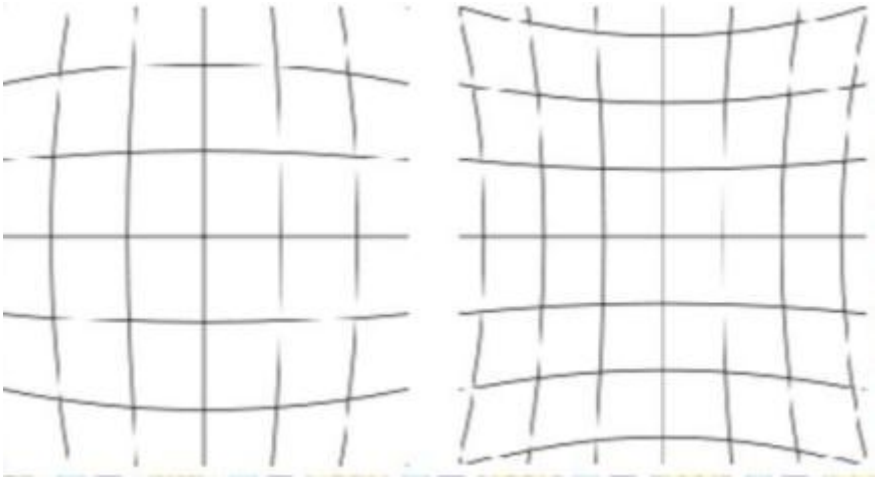
digunakan sebagai marker harus dikonversi terlebih dahulu menjadi file berekstensi *.ass sebelum digunakan didalam kode. Proses konversi gambar menjadi file berektensi *.ass hanya dapat dilakukan dengan melibatkan IN2AR *support team*.

2.8. Kalibrasi Kamera

Ketika sebuah kamera menangkap obyek, sering kali gambar yang dihasilkan tidak sesuai dengan kenyataanya. Beberapa permasalahan ketika menangkap obyek dengan kamera adalah sebagai berikut:

1. Pusat gambar yang tidak terdapat pada bagian tengah
2. Panjang fokus kamera
3. Faktor skala yang berbeda-beda untuk baris piksel
4. Faktor *skew*
5. Distorsi lensa

Distorsi lensa merupakan permasalahan yang sering terjadi karena ketidaksempurnaan lensa. Secara umum distorsi lensa dibagi menjadi dua, yaitu *pin-cushion effect* dan *barrel effect.* Gambar 2.10 menjelaskan bagaimana sebuah *grid* ditangkap oleh lensa tidak sebagaimana mestinya sehingga menghasilkan *pin-cushion effect* dan *barrel effect.*



Gambar 2.5 Kiri: *pin-cushion effect;* Kanan: *barrel effect*

Untuk meminimalisir efek tersebut dan agar mengoptimalkan proses *tracking marker*, dibutuhkan proses kalibrasi kamera. Proses ini dilakukan cukup sekali untuk satu buah perangkat keras kamera. Dengan proses kalibrasi ini, parameter internal dari kamera dapat dikalkulasi sehingga proses kalkulasi informasi tiga dimensi dari suatu gambar menjadi lebih tepat.

2.9. Transformasi Matriks

Dalam pemograman grafis komputer tiga dimensi, sebuah transformasi adalah cara untuk mengambil sebuah titik pada ruang tiga dimensi dan bagaimana cara memanipulasinya. Transformasi dapat dilakukan pada sebuah titik sederhana sampai semua titik tanpa tampak sebuah model geometri yang sangat detail sekalipun. Dalam pemograman grafis, semua transformasi dilakukan dengan menggunakan matriks. Dalam permainan matriks direnderer secara terus menerus maka transformasi yang dilakukan secara berturut-turut. Transformasi berturut-turut dapat dilakukan dengan cara mengalikan matriks-matriks transformasi sesuai urutan transformasi, untuk memudahkan maka matriks transformasi maka dapat

menggunakan *homogeneous transformation,*yaitu dengan mengubah matriks transformasi menjadi ukuran 4x4[4], Seperti pada rumus 2.1. Dalam *library* IN2AR sudah menyediakan fungsi-fungsi yang memudahkan dalam perhitungan matriks. Transformasi memiliki beberapa jenis yaitu transformasi rotasi, translasi dan skala.

$$M = \begin{bmatrix} m00 & m01 & m02 & m03 \\ m10 & m11 & m12 & m13 \\ m20 & m21 & m22 & m23 \\ trx & try & trz & 1 \end{bmatrix} \quad (2.13)$$

2.9.1. Rotasi

Rotasi adalah transformasi untuk memutar suatu titik pada suatu garis sumbu ruang. Kelas matriks pada *actionscript* terdapat fungsi memproduksi matriks rotasi. Parameter yang dibutuhkan untuk membuat rotasi matriks adalah sumbu rotasi dan besarnya sumbu rotasi dalam bentuk radian. Rumus matriks transformasi untuk melakukan rotasi sebesar θ radian

$$M = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\,\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.14)$$

Keterangan:

M : Matriks rotasi

θ : Sudut rotasi dalam radian

2.9.2. Translasi

Translasi adalah jenis transformasi yang digunakan untuk mengubah posisi sebuah titk ke posisi yang lain. Untuk melakukan translasi juga dapat

menggunakan matriks. Rumus 2.14 adalah matriks transformasi untuk melakukan translasi sejauh (*dx*,*dy*,*dz*).

$$M = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.15)$$

Keterangan:

M : Matriks Translasi

dx : Translasi terhadap sumbu x

dy : Translasi terhadap sumbu y

dz : Translasi terhadap sumbu z

2.9.3. Skala

Transformasi skala alaha sebuah tipe transformasi yang digunakan untuk mengubah ukutan suatu obyek. Rumus 2.15 adalah matriks transformasi untuk melakukan skala sebesar (*Sx*,*Sy*,*Sz*).

$$M = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sx & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.16)$$

Keterangan :

M : Matriks Skala

Sx : Skala terhadap sumbu x

Sy : Skala terhadap sumbu y

Sz : Skala terhadap sumbu z [17].

2.10. *Matrix Euler*

Matriks rotasi untuk grafis 3D dengan representasi *euler* terdiri atas rotasi terhadap sumbu x, y, dan z.

Matriks rotasi tehadap sumbu x:

$$Rx(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \tag{2.17}$$

Matriks rotasi tehadap sumbu y:

$$Ry(\beta) = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \tag{2.18}$$

Matriks rotasi terhadap sumbu z:

$$Rz(\gamma) = \begin{bmatrix} cos\gamma & sin\gamma & 0 \\ -sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

Matriks rotasi untuk grafis 3D dengan representasi euler terdiri dari rotasi terhadap sumbu x,y, dan z. Perkalian matriks tidak bersifat komutatif, maka dengan repersentasi *euler* terdapat 12 urutan rotasi yang dapat dihasilkan yaitu : xyz, xzy, xyx, xzx, yxz, yzx, yxy, yzy, zyx, zyx, zxz, zyz. Untuk rotasi zyx, didapatkan rumus sebagai berikut:

$Rx(\alpha)xRy(\beta)xRz(\gamma) =$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} cos\gamma & sin\gamma & 0 \\ -sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

sehingga didapatkan

$$=$$

$$\begin{bmatrix} cos\beta\ cos\gamma & cos\beta\ sin\gamma & -sin\beta \\ sin\alpha\ sin\beta\ cos\gamma - cos\alpha\ sin\gamma & sin\alpha\ sin\beta\ sin\gamma + cos\alpha\ cos\gamma & sin\alpha\ cos\beta \\ cos\alpha\ sin\beta\ cos\gamma + sin\alpha\ sin\gamma & cos\alpha\ sin\beta\ sin\gamma - sin\alpha\ cos\gamma & cos\alpha\ cos\beta \end{bmatrix}$$

(2.21)

Karena menggunakan aturan tangan kanan (*Right-Hand Rule*) maka α, β, γ adalah sudut antara $0 - 360°$, berlawanan arah dengan arah jarum jam [5]. Fungsi matriks *euler* ini sudah terdapat dalam *library Papervision3D*.

2.11. FlashDevelop

FlashDevelop merupakan sebuah editor gratis yang bisa bekerja dalam sistem operasi Windows, maupun Mac OSX, atau pun Linux. Akan tetapi sejak membutuhkan Microsoft .NET 2.0 runtime, FlashDevelop hanya berjalan di Windows. FlashDevelop mendukung koding yang dilakukan oleh ActionScript 3.0.

2.11.1. ActionScript 3.0

ActionScript adalah bahasa pemrograman yang di pakai oleh *software* Flash untuk mengendalikan obyek-obyek ataupun *movie* yang terdapat dalam Flash. ActionScript 3.0 mempunyai beberapa tipe data primitif seperti yang diberikan dalam table 2.2. selain itu actionscript juga mempunyai beberapa tipe data kompleks yang dibangun dari tipe data prmitif seperti yang diberika dalam table 2.2. bahkan kita bisa membangun tipe data sendiri, dalam bahasa pemograman obyek sering dikenal dengan nama *class* obyek.

Table 2.2 Tipe data primitive ActionScript 3.0

Tipe Data	Pengertian	Contoh
String	Sebuah karekter atau sederet	Var textsaya:String =

	karakter	“Assalamu’alaikum” Bisa juga ditulis dengan : Var textsaya: string Textsaya =”Assalamu’alaikum”
Boolean	Hanya bernilai benar(true) atau salah (false)	Var khilafah: boolean = true
Number	Bilangan riil positif dan negative	Var percepatan:number = 0.01
Int	Bilangan bulat positif dan negative (-2.147.483.648 s.d 2.147.483.648)	Var radiusbola : int = 10
Uint	Bilangan bulat positif (0 s.d 4.294.967.295)	Var warnabola : uint = 0xFF00FF
*	Tipe data sembarang (biasanya digunakan untuk menampung data yang belum diketahui tipe datanya)	Var databaru : * = datasembarang

Table 2.3 Beberapa tipe data kompleks ActionScript 3.0

Tipe Data	Pengertian	Contoh
MovieClip	Sebuah symbol movie clip	Var Al-fatih : movieClip = new MovieClip()
Sprite	Sebuah sprite	Var kotak : sprite = new sprite ()
TextField	Sebuah field text yang bertipe dynamic atau field teks yang bertipe input (secara default akan bertipe dynamic)	Var Islam : textFiled = new textField () Islam.text = “Syari’ah dan Khilafah”
SimpleButton	Sebuah symbol button (tombol)	Var start.SimpleButton=New simpleButton()
Date	Data dalam format tanggal	Var lebaran: Date = new Date ()
Array	Larik atau array yang dapat memuat lebih dari satu nilai	Var arrayBilangan: array = new Array () ArrayBilangan = [2,3,15]
Obyek	Struktur dasar dari setiap entitas ActionScript. Obyek juga merupakan suatu bentuk variable yang bisa diatur sendiri untuk menyinpan berbagai nilai sebagai alternative dari array	Var gambar:obyek: New object Gambar.nama = foto action“ Gambar.panjang = 600

ActionScript 3.0 mendefinisikan empat kata untuk menunjukan ruang lingkup variable (ataupun fungsi/method), yaitu :

1. Public : dapat diakses oleh semua bagian kede didalam keseluruhan aplikasi / project tempat variable didefinisikan7
2. Private : hanya dapat diakses oleh method/ fungsi tempat variable didefinisikan
3. Internal : dapt diakses oleh sembarang *class* yang didefinisikan dalam package yang sama.
4. Protected : hanya dapat diakses oleh method/ fungsi tempat variable didefinisikan dan *class* yang extend *class* tempat variable didefinisikan.

Salah satu kelebihan Actionscript 3.0 adalah dukungannya terhadap pemograman menggunakan *class-class*. Dengan penggunan *class*, anda dapat menerapakan konsep pemograman berorientasi obyek dalam pengkodean program. Sebuah *class* adalah cetak biru dari sebuah obyek. *Class* mendefinisikan obyek secara penuh, meliputi data obyek operasi-operasinya. Untuk membuat *class* perlu diperhatikan beberapa hal berikut ini:

a. Pembuatan *class* dalam actionscript harus diawali dengan sebuah *package*. Sebuah packet adalah grup dari *class* (bisa hanya satu *class* atau beberapa *class*). Bisa juga menambahkan nama forlder di belakang kata package yang menunjukan direktori tempat anda menyimpan *class* anda. Misalnya anda bisa membuat *class* Bola

yang akan anda taruh dalam folder lib dengan menambahkan kata lib di belakang kata *package*.

b. Dokumen dari *class* harus disimpan dengan nama yang sama dengan *class*, misalnya nama*Class*.as. Misalnya *class* Bola yang anda buat harus anda simpan dengan nama bola.as.

c. Sebaliknya *class* yang dibuat merujuk ke *class* yang sudah dimiliki Flash, misalnya Sprite atau MovieClip gunakan tambahan kata extends Sprite pada lanjutan nama *class*nya, misalnya public *class* bola *extends Sprite*.

d. Menambahkan variable dengan ruang lingkup *public*, internal ataupun yang lainnya sesuai kebutuhan. Misalnya menambahkan variabel-variabel yang bersifat *public*, seperti : radius yang menunjukan jari-jari bola, warna yang menunjukan warna bola, vx yang menunjukan kecepatan bola searah sumbu-x, dan vy yang menunjukan bola searah sumbu-y.

e. Menambahkan method berupa fungsi dengan ruang lingkup yang diinginkan didalam fungsi. Salah satu fungsi yang sebaiknya ada adalah fungsi dengan nama fungsi sama dengan nama *class*.

2.12. *Papervision3D*

Papervision3D merupakan sebuah *library* yang digunakan dalam pemograman berbahasa actionscript. *Papervision3D* adalah sebuah *open source* 3D yang popular untuk flash, dengan *Papervision3D* memungkinkan pengguna untuk membuat benda tiga dimensi real time yang ditampilkan pada *browser*.

Papervision3D mendukung Actionscript 2 dan 3 serta dirancang untuk format COLLADA yang juga mendukung polygonal geometri, hierarki dan tekstur obyek peta.

2.13. UML(*Unified Modeling Language*)

UML (*Unified Modeling Language*) adalah sebuah bahasa yang berdasarkan grafik/gambar untuk memvisualisasi, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan software berbasis OO (*Object-Oriented*). UML sendiri juga memberikan standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa program yang spesifik, skema database. UML sebagai sebuah bahasa yang memberikan *vocabulary* dan tatanan penulisan kata-kata dalam '*MS Word*' untuk kegunaan komunikasi. Sebuah bahasa model adalah sebuah bahasa yang mempunyai *vocabulary* dan konsep tatanan / aturan penulisan serta secara fisik mempresentasikan dari sebuah sistem. Seperti halnya UML adalah sebuah bahasa standard untuk pengembangan sebuah *software* yang dapat menyampaikan bagaimana membuat dan membentuk model-model, tetapi tidak menyampaikan apa dan kapan model yang seharusnya dibuat yang merupakan salah satu proses implementasi pengembangan *software*.

2.13.1. Bagian-Bagian UML

Bagian-bagian utama dari UML adalah *view*, diagram, *model element*, dan *general mechanism*.

a. *View*

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain: *use case view, logical view, component view, concurrency view,*dan *deployment view*.

b. *Use case view*

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. Aktor yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya. *View* ini digambarkan dalam *use case* diagram dan kadang-kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

c. *Logical view*

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class*, object, dan *relationship*) dan kolaborasi dinamis yang terjadi ketika object mengirim pesan ke object lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam *class* diagrams untuk struktur statis dan dalam *state, sequence, collaboration*, dan *activity diagram* untuk model dinamisnya. *View* ini digunakan untuk perancang (*designer*) dan pengembang (developer).

d. *Component view*

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administratif lainnya. *View* ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

e. *Concurrency view*

Membagi sistem ke dalam proses dan prosesor.*View* ini digambarkan dalam diagram dinamis (*state, sequence, collaboration,* dan *activity diagrams*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

f. *Deployment view*

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

g. Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk *view* tertentu. Adapun jenis diagram antara lain :

1. *Use Case Diagram*

Use case adalah abstraksi dari interaksi antara sistem dan aktor. *Use case* bekerja dengan cara mendeskripsikan tipe interaksi antara *user* sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. *Use case* merupakan konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata *user*. Sedangkan *use case* diagram memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*.

2. *Class Diagram*

Class adalah dekripsi kelompok obyek-obyek dengan property, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class* diagram dapat memberikan pandangan global atas sebuah sistem. Hal tersebut tercermin dari *class-class* yang ada dan relasinya satu dengan yang lainnya. Sebuah sistem biasanya mempunyai beberapa *class* diagram. *Class* diagram sangat membantu dalam visualisasi struktur kelas dari suatu sistem.

3. *Component Diagram*

Component Diagram merupakan bagian fisik dari sebuah sistem, karena menetap di komputer tidak berada di benak para analis. Komponent merupakan implementasi software dari sebuah atau lebih *class*. Komponent dapat berupa *source code*, komponent biner, atau *executable component*. Sebuah komponent berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical*

view ke *component view*.Sehingga *component diagram* merepresentasikan dunia *real* yaitu *component software* yang mengandung *component*, *interface* dan *relationship*.

4. *Deployment Diagram*

Menggambarkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian software yang berjalan pada bagian-bagian *hardware,* menunjukkan hubungan komputer dengan perangkat (nodes) satu sama lain dan jenis hubungannya. Di dalam *nodes, executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen.

5. *State Diagram*

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu object dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa object lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

6. *Sequence Diagram*

Sequence Diagram digunakan untuk menggambarkan perilaku pada sebuah skenario. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara object juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

7. *Collaboration Diagram*

Menggambarkan kolaborasi dinamis seperti sequence diagrams.Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

8. *Activity Diagram*

Menggambarkan rangkaian aliran dari aktifitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

2.14. *Rasional Rose*

Salah satu *tool* pendesainan yang mendukung UML, yaitu *rational rose*. *Rational rose* merupakan salah satu *software* yang paling banyak digunakan untuk melakukan *design software* melalui pendekatan UML (*Unified Modelling Language*). *Rational rose* merupakan *software* yang menyediakan banyak fungsi-fungsi seperti*: design* proses *generate code, reverse engineering*, serta banyak fungsi-fungsi yang lain. *Rational rose* merupakan *tool* yang sangat mudah karena sudah menyediakan contoh-contoh *design* dari beberapa *software*.